ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE INFORMÁTICA Y ELECTRÓNICA

ESCUELA DE INGENIERÍA EN SISTEMAS

"ESTUDIO COMPARATIVO DE LIBRERIAS DE COMPONENTES PARA DESARROLLO DE APLICACIONES WEB CON INTERFACES ENRIQUECIDAS CON JSF, APLICADO AL SISTEMA DE CONTROL VEHICULAR DE LA ESPOCH"

TESIS DE GRADO

PREVIA A LA OBTENCIÓN DEL TÍTULO DE

INGENIERA EN SISTEMAS INFORMÁTICOS

PRESENTADO POR:

EULALIA XIMENA CARRILLO ROBALINO

KARLA MARIUXI SOSA PAREDES

RIOBAMBA – ECUADOR

2013

El presente trabajo de tesis lleva un cordial y fraterno agradecimiento a la Escuela Superior Politécnica de Chimborazo, a la Escuela Ingeniería en Sistemas y al Departamento de Sistemas Y Telemática por habernos permitido desarrollar el presente proyecto de tesis facilitándonos el conocimiento y la tecnología necesaria para la realización de ésta tesis.

Agradeciendo a nuestros maestros y asesores Ing. Landy Ruiz, Ing. Ivonne Rodríguez, Ing. Diego Palacios, Ing. Aníbal Herrera, quienes con sus conocimientos y experiencias nos guiaron de forma correcta para el desarrollo de nuestro proyecto de tesis.

Éste trabajo de tesis lo dedico a mi hija que es la bendición más grande que Dios me ha dado, ella es el motivo de todos mis esfuerzos, el motor que me obliga a funcionar y ser cada día mejor, la fuente de mi inspiración y motivación para culminar con este proyecto y cumplir con mis dos más grandes sueños; ser madre de una preciosura "Lesly Alejandra Quinatoa Carrillo" y mi título de Ingeniera en Sistemas Informáticos.

Eulalia Carrillo Robalino

Dedico ésta tesis con mucho amor y respeto a mi familia en especial a mis padres, Rosalía y Olimpo por sus sacrificios y apoyo incondicional durante el transcurso y culminación de mi carrera profesional.

A mis hermanos Jonathan, Alexander e Ivette quienes estuvieron cuando los necesite.

A mi bello sobrino Justhin quien con su nacimiento llegó a ser la luz en mi vida.

Karla Sosa Paredes

FIRMAS DE RESPONSABILIDAD

	FIRMA	**FECHA**
Ing. Iván Menes **DECANO DE LA FACULTAD** **INFORMÁTICA Y ELECTRÓNICA**	**.....**	**.....**
Ing. Raúl Rosero **DIRECTOR DE LA ESCUELA** **INGENIERÍA EN SISTEMAS**	**.....**	**.....**
Ing. Ivonne Rodríguez **DIRECTORA DE TESIS**	**.....**	**.....**
Ing. Landy Ruiz **MIEMBRO DEL TRIBUNAL**	**.....**	**.....**
Tlgo. Carlos Rodríguez **DIRECTOR CENTRO DE DOCUMENTACIÓN**	**.....**	**.....**

“Nosotras, EULALIA XIMENA CARRILLO ROBALINO y KARLA MARIUXI SOSA PAREDES somos responsables de las ideas, doctrinas y resultados expuestos en esta tesis; y el patrimonio intelectual de la Tesis de Grado pertenece a la ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO”.

EULALIA XIMENA CARRILLO ROBALINO KARLA MARIUXI SOSA PAREDES

INDICE DE ABREVIATURAS

AJAX	Asynchronous JavaScript And XML, Java Script Asíncrono
CSS	Hoja de Estilo en Cascada
DESITEL	Departamento de Sistemas y Telemática
DOM	Modelo de Objetos de Documentos
DPI	Derechos de propiedad intelectual
ESPOCH	Escuela Superior Politécnica de Chimborazo
GUI	Interfaz Gráfica de Usuario
HTML	Hiper Text Markup Language
JSF	JavaServer Faces
JSP	JavaServer Pages
MVC	Modelo Vista Controlador
RIA	Rich Internet Application
SCJM	Sistema Control Jefatura Movilización
XP	eXtreme Programming

Tabla de contenido

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Al afrontar cualquier proyecto de diseño y desarrollo de páginas web para su integración en un marco de actuación como es Internet, se hace preciso contar con un conocimiento y control precisos sobre herramientas y medios de alta sofisticación, si se quiere tener una presencia relevante, productiva y orientada a satisfacer todos y cada uno de los objetivos previstos en tan dinámico y estimulante medio.

No se debe tomar en cuenta los proyectos como simples aplicaciones de diseño convencional, se requiere tener en cuenta la navegabilidad, interactividad, usabilidad, arquitectura de la información y la interacción de medios como el audio, texto, imagen, enlaces y vídeo.

Existen varias herramientas de aplicaciones web enriquecidas de las cuales se nos hace difícil escoger la adecuada para el desarrollo de las aplicaciones.

Se tomó como escenario a la Unidad de Movilización de la Escuela Superior Politécnica del Chimborazo, donde se ha visto la necesidad de implementar una aplicación web que permita la administración y control del sistema vehicular.

Para la automatización de este proceso se desarrollará una solución informática de tal manera que pueda ofrecer un correcto tratamiento de la información brindando un mejor desenvolvimiento de la Unidad de Movilización.

CAPÍTULO I

Marco referencial

1.1. Antecedentes

A pesar de que los años pasan y la tecnología avanza a pasos agigantados, todavía se ven muchos sitios web y aplicaciones que desarrollan sus interfaces para que “se vean bonitas” en vez de pensar en cosas como la usabilidad y la accesibilidad.

Para el caso particular de las aplicaciones con interfaces web, se las puede dividir en dos partes muy marcadas: el lado del servidor, donde se realiza la programación dura, en lenguajes como PHP, Java, Python, etc., y el lado del cliente, en donde se utilizan (X) HTML, CSS Javascript para brindar los contenidos a los usuarios.

Se puede desarrollar páginas web con tablas que se van a ver realmente bien, o se puede llenar de código sucio una página que los visitantes en su mayoría no se van a dar cuenta. Pero no solo se debe desarrollar para obtener algo bonito, sino además algo funcional, accesible, fácil de mantener, rápido de cargar y profesional.

Además que en el momento de desarrollar, los programadores realizan las páginas web en JSP (JavaServer Page) de manera sencilla pero se tiene que codificar casi todo perdiendo tiempo.

La tecnología *JavaServer Faces* constituye un marco de trabajo de interfaces de usuario del lado de servidor para aplicaciones web basadas en tecnología Java y en el patrón MVC (Modelo Vista Controlador), usando librerías de componentes para un desarrollo más ágil y rápido.

PrimeFaces es una librería de componentes visuales open source desarrollada y mantenida por Prime Technology, una compañía Turca de IT especializada en consultoría ágil, JSF, Java EE y Outsourcing.

IceFaces permite al programador incluir una serie de tags Ajax en sus JSP o Xhtml de tal manera que el Ajax es generado por el framework de manera automática.

RichFaces es un marco muy útil de código abierto que permite añadir capacidades de Ajax a sus aplicaciones JSF, usando los componentes estándar JSF, sin la necesidad de escribir código JavaScript y administrar la compatibilidad de JavaScript entre navegadores.

Apache MyFaces es un proyecto de Apache Software Foundation, para mantener una implementación abierta de JavaServer Faces JSF, por medio del desarrollo de bibliotecas y componentes.

OpenFaces es una librería open-source de AJAX potencia componentes JSF, un framework Ajax y un marco de validación en el cliente, se basa en el conjunto de componentes JSF antes conocido como QuipuKit. Contiene código base completamente revisada de QuipuKit e introduce muchos nuevos componentes y características.

Así, nace la necesidad de realizar este estudio comparativo entre componentes innovadores y eficaces para el desarrollo de aplicaciones web con JavaServer Faces, determinando los pros y contras de cada una de estos componentes y así brindar al desarrollador una guía al momento de elegir uno de ellos.

Desarrollar criterios de diseño que permitan seleccionar la tecnología a utilizar según el tamaño del sistema a desarrollar, el tipo de plataforma sobre el cual debe funcionar, la complejidad del negocio que el sistema resuelve, etc., son criterios que se deben contemplar, también la funcionalidad capturada en el análisis del problema que el sistema en desarrollo busca resolver.

Actualmente existen entidades públicas y privadas que poseen importante información que es manejada de forma manual, dicho manejo muchas veces se hace de forma inadecuada, con lo cual se genera información falsa o adulterada que solo tiende a aumentar el desconcierto y la falta de transparencia.

La información se convierte en una herramienta estratégica y en un activo de gran valor, conlleva a que la gestión de seguridad de la información se convierta también en un tema de gran relevancia para todas las entidades.

Es por eso que con la ayuda de herramientas informáticas se facilitará el análisis y la posterior toma de decisiones.

En la Unidad de Movilización de la Escuela Superior Politécnica del Chimborazo se ha visto la necesidad de implementar una aplicación web que permita la administración y control del sistema vehicular, la misma que estará ligada al Sistema Financiero de dicha entidad educativa.

Se requiere analizar y transparentar información de los diferentes trámites y órdenes para el control vehicular y varias operaciones que actualmente se construye manualmente lo que conlleva a un trabajo laborioso, tedioso y requiere mucho tiempo.

Para la automatización de este proceso desarrollaremos una solución informática de tal manera que pueda ofrecer un correcto tratamiento de la información brindando un mejor desenvolvimiento de la Unidad de Movilización.

1.2. Justificación del tema

1.2.1. Justificación teórica

Hoy en día se cuenta con las herramientas para la creación de aplicaciones web, las mismas que ayudan al desarrollador a realizar este tipo de aplicaciones de una manera rápida y sencilla, estas fueron creadas para minimizar el tiempo de desarrollo de un sistema de aplicación web.

JSF, JavaServer Faces, es un lenguaje orientado a la creación de árboles de componentes visuales en el servidor de modo que trata de independizar el desarrollo de la interfaz visual,

del lenguaje del cliente en el que se interpretará. JSF nos proporciona la renderización de los componentes visuales en el lenguaje del cliente, no debemos preocuparnos del html o javascript que se necesitará en el cliente para generarlo, puesto que ya lo incorpora el renderizador del propio componente.

El desarrollo de sistemas basado en componentes, es una aproximación del desarrollo de software que describe, construye y utiliza técnicas de software para la elaboración de sistemas abiertos y distribuidos mediante el ensamblaje de partes software reutilizables. La aproximación del desarrollo de sistemas basado en componentes es utilizada para reducir los costes, tiempos y esfuerzos de desarrollo del software, a la vez que ayuda a mejorar la fiabilidad, flexibilidad y la reutilización de la aplicación final.

Mediante el uso de componentes visuales de JSF, se logrará crear la aplicación de forma simple, encapsulando la mayoría de la lógica en componentes visuales, que generen editores de texto, lista de valores dinámicas, o campos de texto con funciones de auto completar texto.

IceFaces es un Framework Ajax que permite desarrollar Aplicaciones RIA (Aplicaciones de Internet Enriquecidas).

RichFaces es una biblioteca de código abierto basada en Java usando librerías de componentes que permite crear aplicaciones web con Ajax.

PrimeFaces es un framework JSF, una librería de componentes visuales open source.

Apache MyFaces establece varias bibliotecas de componentes que contienen los widgets de interfaz de usuario para la construcción de aplicaciones web con JSF.

OpenFaces ofrece una amplia gama de componentes web para crear interfaz web, es open source, utiliza Ajax.

A raíz del estudio comparativo entre las librerías de componentes visuales se logrará determinar cuál de estas nos dará un mejor rendimiento, disponibilidad de componentes, inicio rápido, problemas abiertos, sus características principales para poder desarrollar aplicaciones web.

1.2.2. Justificación práctica

Hoy en día en la Unidad de Movilización la información es manejada de forma manual, dicho manejo muchas veces se hace inadecuadamente, con lo cual se genera información falsa o adulterada que solo tiende a aumentar el desconcierto y la falta de transparencia.

El sistema a desarrollarse consiste en el control vehicular de la Escuela Superior Politécnica de Chimborazo, la cual permitirá la administración y control del mismo teniendo en cuenta que mucha de la información manejada está basada en peticiones gubernamentales como Contraloría y Auditoría Interna, esta además ligado al Sistema Financiero de la ESPOCH.

Para el desarrollo del sistema se procederá a:

1. Crear un prototipo del sistema con las librerías de componentes más adecuadas en el desarrollo de aplicaciones web enriquecidas para determinar cuál de estas brinda mayores beneficios en tiempo de desarrollo del sistema de control vehicular de la ESPOCH.
2. El sistema de control vehicular estaría compuesto de los siguientes módulos:

- **Módulo de Gestión Vehicular**.- Este módulo permitirá ingresar, actualizar, eliminar un vehículo además de controlar las fechas del Soat y de la Matricula con su respectiva información.
- **Módulo Despacho de Gasolina**.- Este módulo permitirá la administración y control del proceso de tanqueado de gasolina a los vehículos.
- **Módulo de Reportes**.- Este módulo permitirá generar los reportes mensuales, trimestrales y anuales que el jefe de movilización debe hacer al Vicerrectorado de Investigación y Desarrollo.
- **Integración de Seguridad y Usuarios al Sistema.**

1.3. Objetivos

1.3.1. Objetivo general

Realizar un estudio comparativo de librerías de componentes para desarrollo de aplicaciones web con interfaces enriquecidas con JSF, aplicado al Sistema de Control Vehicular de la ESPOCH.

1.3.2. Objetivos específicos

- Determinar parámetros de comparación entre las librerías componentes para desarrollo de aplicaciones web con interfaces enriquecidas con JSF.
- Seleccionar las librerías de componentes más adecuadas para el desarrollo de aplicaciones web enriquecidas.
- Construir prototipos para realizar el análisis comparativo entre las librerías seleccionadas.
- Diseñar e implementar una aplicación Web para el sistema de control vehicular de la Escuela Superior Politécnica el Chimborazo utilizando la librería de componentes seleccionada.

1.4. Hipótesis

El uso de librerías de componentes adecuadas para el desarrollo de aplicaciones web enriquecidas reduce el tamaño de página y el tamaño de una respuesta Ajax.

CAPÍTULO II

Marco teórico

JavaServer Faces es un framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE.

La tecnología Ajax actualmente viene inherente en la arquitectura del framework JSF, esto quiere decir que el desarrollador no debe programar nada extra para que esta tecnología se aplique.

De la misma manera existe una diversidad de librerías de componentes para JSF que ayudan al desarrollador a efectuar aplicaciones web enriquecidas de manera rápida y eficiente.

Todos estos temas serán tratados en el transcurso del presente capítulo, definiendo conceptos, características, ventajas, desventajas, etiquetas de desarrollo para con lo cual tener un conocimiento más amplio de éste framework y sus librerías.

2.1. JavaServer faces

Hoy el día a día es trabajar para mejorar la calidad del software, para esto se considera fundamental el uso de patrones. Uno de los patrones más conocidos en el desarrollo web es

el patrón MVC Modelo Vista Controlador (Figura 1), que permite separar la lógica de control (saber cosas hay que hacer pero no como), la lógica de negocio (saber cómo se hacen las cosas) y la lógica de presentación (saber cómo interactuar con el usuario).

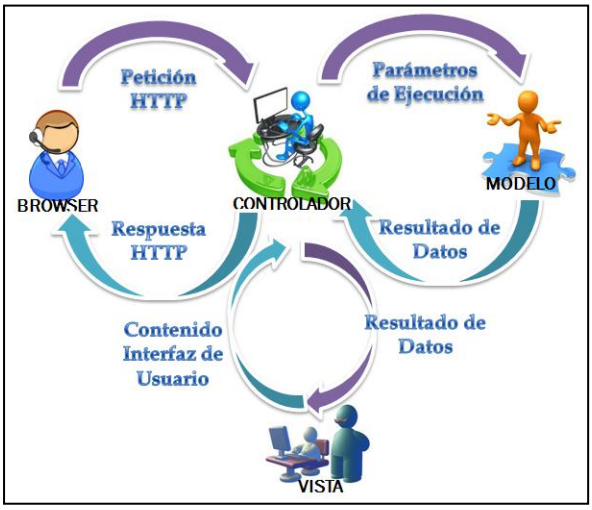


Figura II. 1. MVC. Modelo Vista Controlador

Utilizando este tipo de patrones se gana: más calidad, mejor mantenibilidad, pero una de las cosas más importantes es la normalización y estandarización del desarrollo de Software.

Tradicionalmente, las aplicaciones web se han codificado mediante páginas JSP, *JavaServer Pages*, que recibían peticiones a través de formularios y construían como respuesta páginas HTML, *Hiper Text Markup Language*, mediante ejecución directa o indirecta a través de bibliotecas de etiquetas de código Java.

2.1.1. ¿Qué es JavaServer faces?

JavaServer Faces (JSF), Figura 2, es un framework estándar Java que permite construir aplicaciones Web. Se define por las especificaciones JSR 127, JSR 252 y JSR 276 basado

en el patrón MVC, que pretende normalizar y estandarizar el desarrollo de aplicaciones web.

Figura II. 2. JSF – JavaServer Faces

La tecnología JavaServer Faces es un marco de trabajo de interfaces de usuario del lado de servidor para aplicaciones Web.

JSF es un conjunto de componentes de usuario que permite construir la capa de vista o presentación de las aplicaciones Web proporcionando un conjunto de APIs para desarrollar estos componentes GUI y gestionar su funcionamiento mediante el tratamiento de eventos, las validaciones de entrada, la definición de la navegación entre páginas y el soporte de accesibilidad. Las clases de componentes GUI incluidas en JSF encapsulan su funcionalidad y no la presentación en un tipo de cliente específico por lo que esto permite poder ser mostrados en distintos clientes.

JSF incluye:

- Un conjunto de APIs para representar componentes de una interfaz de usuario y administrar su estado, manejar eventos, validar entrada, definir un esquema de navegación de las páginas y dar soporte para internacionalización y accesibilidad.
- Un conjunto por defecto de componentes para la interfaz de usuario.

- Dos librerías de etiquetas personalizadas para JSP que permiten expresar una interfaz JSF dentro de una página JSP.
- Un modelo de eventos en el lado del servidor
- Administración de estados.
- Beans administrados

Se compone por librerías que residen en el lado del servidor y contienen todos los elementos del framework, incluyendo el conjunto de componentes que se despliegan en las páginas XHTML.

JSF implementa el patrón MVC, lo que separa la lógica de navegación de las páginas, en clases Java llamadas Controllers o Controladores, los cuales pueden ser programados utilizando solo anotaciones.

Tiene como estándar de presentación el lenguaje de marcado XHTML, que es una extensión de la tecnología XML y HTML.

La actualización de secciones en las páginas, está controlado por el atributo *render* que contienen la mayoría de los componentes del framework. Este atributo indica que sección debe ser actualizada después de una petición al servidor.

Como se puede apreciar en la figura 3, la interface de usuario que se crea con la tecnología JSF, *miUI*, se ejecuta en el servidor y se renderiza en el cliente.

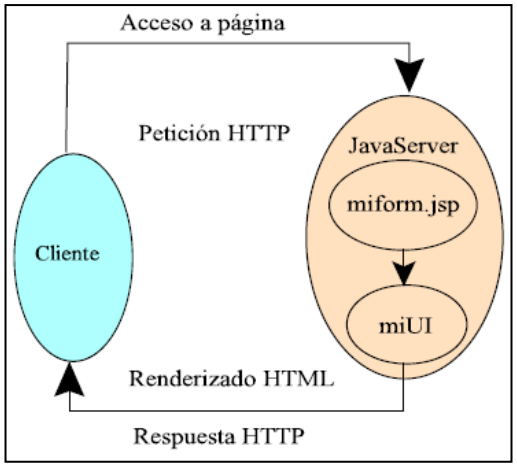


Figura II. 3. Diagrama de una aplicación JSF

La página JSP, *miform.jsp*, dibuja los componentes del interface de usuario con etiquetas personalizadas definidas por la tecnología JSF. El UI de la aplicación Web maneja los objetos referenciados por la página JSP:

- Los objetos componentes que mapean las etiquetas sobre la página JSP.
- Los oyentes de eventos, validadores, y los conversores que están registrados en los componentes.
- Los objetos del modelo que encapsulan los datos y las funcionalidades de los componentes específicos de la aplicación.

2.1.2. Ciclo de vida de una petición jsf

El ciclo de vida de una página JavaServer Faces page es similar al de una página JSP, como se muestra en la figura 4:

El cliente hace una petición HTTP de la página y el servidor responde con la página traducida a HTML.

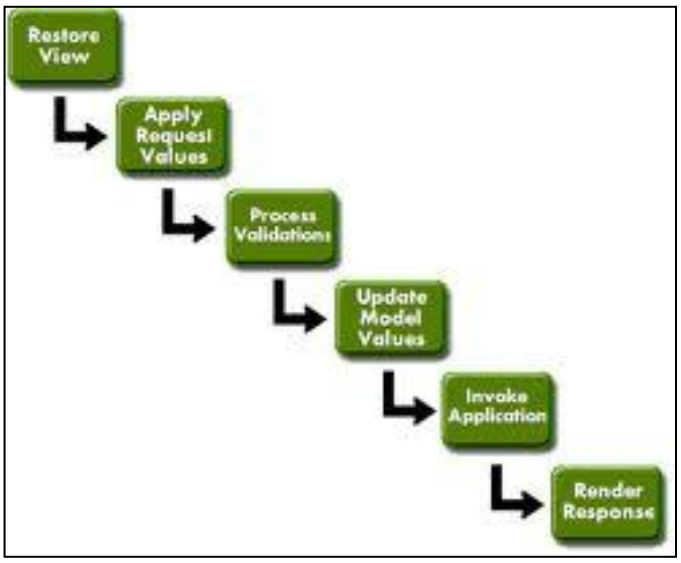


Figura II. 4. Ciclo de Vida de JavaServer Faces

El ciclo de vida completo es:

1. Restaurar vista
2. Aplicar valores de la petición
3. Procesar validaciones
4. Actualizar modelo
5. Invocar aplicación
6. Producir respuesta

Tres escenarios posibles. Cada escenario pasa por distintas fases del ciclo de vida:

- Petición JSF genera respuesta JSF:
    - Ciclo de vida completo
- Petición no-JSF genera respuesta JSF:
    - Restaurar vista
    - Producir respuesta
- Petición JSF genera respuesta no-JSF (ejemplo: generar XML)
    - Restaurar vista

- Aplicar valores de la petición
- Procesar validaciones
- Actualizar modelo
- Invocar aplicación
- Desvío a productor no-JSF

Fase Restaurar vista – RESTORE VIEW. En esta fase se crea el árbol de componentes. Se puede crear a partir de información existente o de cero si no había información. Si no se encuentran datos POST o query string, se pasa directamente a Producir respuesta.

Fase Aplicar valores de la petición – APPLY REQUEST VALUES. Se itera sobre los componentes del árbol, comprobando qué valor de la petición pertenece a qué componente, y los van guardando. Estos valores almacenados se llaman valores locales.

Fase Procesar validaciones – PROCESS VALIDATION. Se realizan las validaciones y conversiones necesarias de los valores locales. Si ocurre algún error en esta fase, se pasa a la fase Render Response, mostrándole al usuario otra vez la página actual y dándole así una nueva oportunidad para que pueda introducir los datos correctos.

Fase Actualizar modelo – UPDATE MODEL VALUES. Se modifican los valores de los beans asociados a los componentes de la vista con los valores locales.

Fase Invocar aplicación – INVOKE APPLICATION. Se invoca el método asociado al action del botón o link y se llevan a cabo las operaciones correspondientes a las acciones del usuario.

Fase Producir respuesta – RENDER RESPONSE. El servidor devuelve la página de respuesta al navegador del usuario y guarda el estado actual de la vista para poder restaurarla en una petición posterior.

2.1.3. Objetivos de JavaServer faces

El objetivo de la tecnología JSF es desarrollar aplicaciones web de forma parecida a como se construyen aplicaciones locales con Java Swing, AWT (*Abstract Window Toolkit*), SWT (*Standard Widget Toolkit*) o cualquier otra API similar. JSF pretende facilitar la construcción de estas aplicaciones proporcionando un entorno de trabajo vía web que gestiona las acciones producidas por el usuario en su página HTML y las traduce a eventos que son enviados al servidor con el objetivo de regenerar la página original y reflejar los cambios pertinentes provocados por dichas acciones.

Estos objetivos de diseño representan el foco de desarrollo de JSF:

- Definir un conjunto simple de clases base de Java para componentes de la interfaz de usuario, estado de los componentes y eventos de entrada.
- Proporcionar un conjunto de componentes para la interfaz de usuario, incluyendo los elementos estándares de HTML para representar un formulario.
- Proporcionar un modelo de JavaBeans para enviar eventos desde los controles de la interfaz de usuario del lado del cliente a la aplicación del servidor.
- Definir APIs para la validación de entrada, incluyendo suporte para la validación en el lado del cliente.
- Especificar un modelo para la internacionalización y localización de la interfaz de usuario.

- Automatizar la generación de salidas apropiadas para el objetivo del cliente, teniendo en cuenta todos los datos de configuración disponibles del cliente, como versión del navegador.

2.1.4. ¿Qué es una aplicación JavaServer faces?

En su mayoría, las aplicaciones JavaServer Faces son como cualquier otra aplicación Web Java. Se ejecutan en un contenedor Servlet Java, y típicamente contienen:

- Componentes JavaBeans, llamados objetos del modelo en tecnología JSF, conteniendo datos y funcionalidades específicas de la aplicación.
- Oyentes de Eventos.
- Páginas, cómo páginas JSP.
- Clases de utilidad del lado del servidor, como *Beans* para acceder a las bases de datos.

Además de estos ítems, una aplicación JavaServer Faces también tiene:

- Una librería de etiquetas personalizadas para dibujar componentes UI en una página.
- Una librería de etiquetas personalizadas para representar manejadores de eventos, validadores, y otras acciones.
- Componentes UI representados como objetos con estado en el servidor.
- Validadores, manejadores de eventos y manejadores de navegación.

Toda aplicación JSF debe incluir una librería de etiquetas personalizadas que define las etiquetas que representan componentes UI y una librería de etiquetas para representar otras acciones importantes, como validadores y manejadores de eventos. La implementación de JSF proporciona estas dos librerías.

La librería de etiquetas de componentes elimina la necesidad de codificar componentes UI en HTML u otro lenguaje de marcas, resultando en componentes completamente

reutilizables, puede ser la librería *html_basic* incluida con la implementación de referencia de la tecnología JavaServer Faces, Y, la librería core hace fácil registrar eventos, validadores y otras acciones de los componentes.

Los componentes UI JavaServer Faces son elementos configurables y reutilizables que componen el interface de usuario de las aplicaciones JavaServer Faces.

La tecnología JavaServer Faces proporciona una arquitectura de componentes rica y flexible que incluye:

- Un conjunto de clases UIComponent para especificar el estado y comportamiento de componentes UI.
- Un modelo de renderizado que define cómo renderizar los componentes de diferentes formas.
- Un modelo de eventos y oyentes que define cómo manejar los eventos de los componentes.
- Un modelo de conversión que define cómo conectar conversores de datos a un componente.
- Un modelo de validación que define cómo registrar validadores con un componente.

La tecnología JavaServer Faces proporciona un conjunto de clases de componentes UI, que especifican toda la funcionalidad del componente, cómo mantener su estado, mantener una referencia a objetos del modelo, y dirigir el manejo de eventos y el renderizado para un conjunto de componentes estándar.

Todas las clases de componentes UI de JavaServer Faces descienden de la clase *UIComponentBase*, que define el estado y el comportamiento por defecto de un

UIComponent. El conjunto de clases de componentes UI incluido en la última versión de JavaServer Faces es:

- UICommand: Representa un control que dispara actions cuando se activa.
- UIForm: Encapsula un grupo de controles que envían datos de la aplicación. Este componente es análogo a la etiqueta **form** de HTML.
- UIGraphic: Muestra una imagen.
- UIInput: Toma datos de entrada del usuario. Esta clase es una subclase de UIOutput.
- UIOutput: Muestra la salida de datos en un página.
- UIPanel: Muestra una tabla.
- UIParameter: Representa la sustitución de parámetros.
- UISelectItem: Representa un sólo ítem de un conjunto de ítems.
- UISelectItems: Representa un conjunto completo de ítems.
- UISelectBoolean: Permite a un usuario seleccionar un valor booleano en un control, selececcionándolo o deseleccionándolo. Esta clase es una subclase de UIInput.
- UISelectMany: Permite al usuario seleccionar varios ítems de un grupo de ítems. Esta clase es una subclase de UIInput.
- UISelectOne: Permite al usuario seleccionar un ítem de un grupo de ítems. Esta clase es una subclase de UIInput.

La arquitectura de componentes JavaServer Faces está diseñada para que la funcionalidad de los componentes se defina mediante las clases de componentes, mientras que el renderizado de los componentes se puede definir mediante un renderizador separado. Este diseño tiene varios beneficios:

- Se puede definir sólo una vez el comportamiento de un componente, pero se pueden crear varios renderizadores, cada uno de los cuales define una forma diferente de dibujar el componente para el mismo cliente o para diferentes clientes.
- Los autores de páginas y los desarrolladores de aplicaciones pueden modificar la apariencia de un componente de la página seleccionando la etiqueta que representa la combinación componente/renderizador apropiada.

Un kit renderizador define como se mapean las clases de los componentes a las etiquetas de componentes apropiadas para un cliente particular. La implementación JavaServer Faces incluye un *RenderKit* estándar para renderizar a un cliente HTML.

Cada etiqueta JSF personalizada en el *RenderKit* de HTML, está compuesta por la funcionalidad del componente, definida en la clase *UIComponent*, y los atributos de renderizado, definidos por el *Renderer*.

La implementación de referencia de JavaServer Faces proporciona una librería de etiquetas personalizadas para renderizar componentes en HTML. Soporta todos los componentes listados en la siguiente tabla:

Tabla II. I. Etiquetas personalizadas para renderizar componentes en HTML

ETIQUETA	DESCRIPCIÓN	SE RENDERIZA COMO
command_button	Enviar un formulario a la aplicación	Un elemento <input type=type> HTML, donde el valor del tipo puede ser submit, reset, o image.
command_hyperlink	Enlaza a otra página o localización en otra página	Un elemento <a href> HTML.
Form	Representa un formulario de entrada. Las etiquetas internas del formulario reciben los datos que serán enviados con el formulario.	Un elemento <form> HTML.
graphic_image	Muestra una imagen	Un elemento <img> HTML.
input_date	Permite al usuario introducir una fecha	Un elemento <input type=text> HTML.
input_datetime	Permite al usuario introducir una fecha y una hora.	Un elemento <input type=text> HTML.
input_hidden	Permite introducir una variable oculta en una página.	Un elemento <input type=hidden> HTML.
input_number	Permite al usuario introducir un número.	Un elemento <input type=text> HTML.
input_secret	Permite al usuario introducir un string sin que aparezca el string real en el campo.	Un elemento <input type=password> HTML.
input_text	Permite al usuario introducir un string.	Un elemento <input type=text> HTML.
input_textarea	Permite al usuario introducir un texto multi-líneas.	Un elemento <textarea> HTML.
input_time	Permite al usuario introducir una hora.	Un elemento <input type=text> HTML.
output_date	Muestra una fecha formateada.	Texto normal.
output_datetime	Muestra una fecha y hora formateados.	Texto normal.
output_errors	Muestra mensajes de error.	Texto normal
output_label	Muestra un componente anidado como una etiqueta para un campo de texto especificado.	Un elemento <label> HTML.
output_message	Muestra un mensaje localizado	Texto normal.
output_number	Muestra un número formateado.	Texto normal.
output_text	Muestra una línea de texto.	Texto normal.
output_time	Muestra una hora formateada.	Texto normal.
panel_data	Itera sobre una colección de datos.	
panel_grid	Muestra una tabla.	Un elemento <label> HTML. con elementos <tr> y %lt;td>
panel_group	Agrupa un conjunto de paneles bajo un padre.	
panel_list	Muestra una tabla de datos que vienen de una collection, un array, un iterator, o un map.	Un elemento <table> HTML. con elementos <tr> y %lt;td>
selectboolean_checkbox	Permite al usuario cambiar el valor de una elección booleana.	Un elemento <input type=checkbox> HTML.

Selectitem	Representa un ítem de una lista de ítems en un componente UISelectOne.	Un elemento <option> HTML.
Selectitems	Representa una lista de ítems en un componente UISelectOne.	Un elemento <option> HTML.
selectmany_checkboxlist	Muestra un conjunto de checkbox, en los que el usuario puede seleccionar varios.	Un conjunto de elementos <input> HTML.
selectmany_listbox	Permite a un usuario seleccionar varios ítems de un conjunto de ítems, todos mostrados a la vez.	Un conjunto de elementos <select> HTML.
selectmany_menu	Permite al usuario selecionar varios ítems de un grupo de ítems.	Un conjunto de elementos <select> HTML.
selectone_listbox	Permite al usuario seleccionar un ítem de un grupo de ítems.	Un conjunto de elementos <select> HTML.
selectone_menu	Permite al usuario seleccionar un ítem de un grupo de ítems.	Un conjunto de elementos <select> HTML.
selectone_radio	Permite al usuario seleccionar un ítem de un grupo de ítems.	Un conjunto de elementos <input type=radio> HTML.

Fuente: http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/130
Realizado por: Karla Sosa Paredes, Eulalia Carrillo Robalino

En algunas situaciones, se podría querer convertir un dato de un componente a un tipo no soportado por el renderizador del componente. Para facilitar esto, la tecnología JavaServer Faces incluye un conjunto de implementaciones estándar de Converter que permite crear conversores personalizados. La implementación de Converter convierte los datos del componente entre las dos vistas.

Un objetivo de la especificación JavaServer Faces es mejorar los modelos y paradigmas existentes para que los desarrolladores se puedan familiarizar rápidamente con el uso del mismo en sus aplicaciones. En este espíritu, el modelo de eventos y oyentes de JavaServer Faces mejora el diseño del modelo de eventos de JavaBeans, que es familiar para los desarrolladores de GUI y de aplicaciones Web.

La tecnología JavaServer Faces soporta un mecanismo para validar el dato local de un componente durante la fase del **Proceso de Validación**, antes de actualizar los datos del objeto modelo. Al igual que el modelo de conversión, el modelo de validación define un conjunto de clases estándar para realizar chequeos de validación comunes.

2.1.5. Beans y páginas JSF

Las aplicaciones web correctamente planificadas tienen dos partes: la parte de presentación y la lógica de negocio.

La *parte de presentación* afecta a la apariencia de la aplicación, y en el contexto de una aplicación basada en navegadores, la apariencia está determinada por las etiquetas HTML, esto comprende marcos, tipos de caracteres, imágenes, etc.

La *lógica de negocio* se implementa en Java y determina el comportamiento de la aplicación.

En el contexto de JSF, la lógica de negocio está contenida en los *beans*, y el diseño está contenido en las páginas.

Beans

Un *bean* es una clase Java que contiene atributos. Un atributo es un valor identificado por un nombre, pertenece a un tipo determinado y puede ser leído y/o escrito sólo a través de métodos a tal efecto llamados métodos *getter* y *setter*.

En una aplicación JSF, se deben usar *beans* para todos los datos accedidos por una página. Los *beans* son los conductos entre la interfaz de usuario y la trastienda de la aplicación.

Páginas JSF

Se necesita una página JSF por cada pantalla de presentación. Dependiendo de lo que se quiera hacer, las páginas típicas son **.jsp** o **.jsf**.

La página **index.jsf** comienza con las declaraciones de las etiquetas, en la que se aprecia la importación de la librería de etiquetas *core*, se usan, entre otras aplicaciones, para manejo de eventos, atributos, conversión de datos, validadores, recursos y definición de la página, las cuales usan el prefijo **f**, por ejemplo **f:view**, y la librería de etiquetas *html_basic*, su utilización es básicamente para la construcción de formularios y demás elementos de interfaz de usuario, que usa el prefijo **h**, por ejemplo **h:form**:

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
```

En el momento de presentar la página, el método **getAtributo** es invocado para obtener el valor del atributo. Y cuando la página se acepta pulsando el botón **Aceptar**, el método **setAtributo** es invocado estableciendo el valor de usuario que ha entrado.

Tiene un atributo denominado *action*, cuyo valor es usado para activar una regla de navegación, la cual está recogida en el fichero **faces-config.xml**, de manera que al producirse esta acción al pulsar el botón, navegaremos a una nueva página.

2.1.6. Ventajas

- JSF es muy flexible al momento de crear renderizadores para mostrar los componentes en la forma que más convenga.
- Permite construir aplicaciones web que introducen realmente una separación entre el comportamiento y la presentación, separación sólo ofrecida tradicionalmente por arquitecturas UI del lado del cliente.
- Proporciona una rica arquitectura para manejar el estado de los componentes, procesar los datos, validar la entrada del usuario, y manejar eventos.
- Gran cantidad de componentes OpenSource.
- Rápida adaptación.
- Validación automática de datos.
- Proporciona utilidades de conversión de datos.
- Los componentes UI de la página están representados en el servidor como objetos con estado. Esto permite a la aplicación manipular el estado del componente y conectar los eventos generados por el cliente a código en el lado del servidor.

2.1.7. Desventajas

- Sitios públicos en la web que han sido creados utilizando este framework.
- Varias empresas que han acordado ser usuarias JSF
- Cualquier cliente de JavaStudioCreator, Oracle JDeveloper
- Lenta evolución

2.2. Ajax

En las aplicaciones web tradicionales, las acciones del usuario en la página desencadenan llamadas al servidor. Una vez procesada la petición del usuario, el servidor devuelve una nueva página HTML al navegador del usuario.

Ajax se ha convertido en una de las técnicas que más generan interactividad en los sitios y aplicaciones web de hoy en día.

2.2.1. ¿Qué es Ajax?

Es una técnica de desarrollo web para crear aplicaciones interactivas o RIA. Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones.

Ajax es una tecnología asíncrona, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página.

Ajax es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores, dado que está basado en estándares abiertos como JavaScript y DOM.

Ajax no es una tecnología en sí mismo. En realidad, se trata de varias tecnologías independientes que se unen de formas nuevas y sorprendentes, como se muestra en la Figura 5:

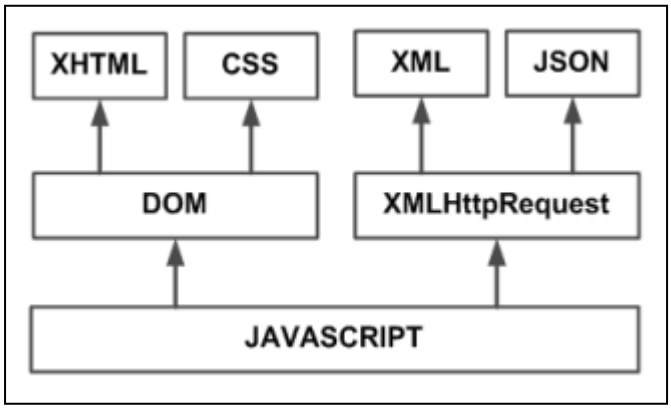


Figura II. 5. Tecnologías agrupadas bajo el concepto de AJAX

- XHTML o HTML y hojas de estilos en cascada (CSS) para el diseño que acompaña a la información.
- DOM accedido con un lenguaje de scripting por parte del usuario, para mostrar e interactuar dinámicamente con la información presentada.
- El objeto XMLHttpRequest para intercambiar datos de forma asíncrona con el servidor web.
- XML es el formato usado generalmente para la transferencia de datos solicitados al servidor.
- JavaScript, para unir todas las demás tecnologías.

2.2.1. Ventajas

- Utiliza tecnologías ya existentes.
- Soportada por la mayoría de los navegadores modernos.
- Interactividad. El usuario no tiene que esperar hasta que lleguen los datos del servidor.
- Portabilidad, no requiere plug-in como Flash y Applet de Java
- Mayor velocidad, esto debido que no hay que retornar toda la página nuevamente.
- La página se asemeja a una aplicación de escritorio.

2.2.2. Desventajas

- Dependiendo de la carga del servidor podemos experimentar tiempos tardíos de respuesta que desconciertan al visitante.
- Las páginas con AJAX son más difíciles de desarrollar que las páginas estáticas.
- Hay problemas usando Ajax entre nombres de dominios. Eso es una función de seguridad.
- Es posible que páginas con Ajax no puedan funcionar en teléfonos móviles, PDA u otros aparatos. Ajax no es compatible con todos los softwares para ciegos u otras discapacidades.

2.3. Librería de componentes

Los componentes permiten al usuario interactuar con la aplicación y proporcionar información desde el programa al usuario sobre el estado del programa. Ejemplos de componentes son: los botones, las barras de desplazamiento, las etiquetas, las listas, las cajas de selección o los campos de texto, entre otros. Cabe mencionar que los componentes nunca se encuentran de forma aislada, sino agrupados dentro de **contenedores o librerías.**

Las librerías contienen y organizan las situaciones de los Componentes; además, son en sí mismos Componentes y como tales pueden ser situados dentro de otros contenedores, los cuales buscan que la aplicación sea entregada en el menor tiempo posible para asegurar su usabilidad, durabilidad y evolución continua de acuerdo a las necesidades, algunos de estos componentes se definirán a continuación:

2.3.1. Richfaces

Richfaces, Figura 6, es una biblioteca de componentes para JSF y un avanzado framework para la integración de AJAX con facilidad en la capacidad de desarrollo de aplicaciones de negocio. Construye sobre el framework de JSF, implementando unos filtros para permitir peticiones Ajax en la página. La singularidad del planteamiento que ofrece es que la petición Ajax provoca una ejecución en el servidor y finalmente una renderización parcial o total de la página del navegador. Así, el control de lo que sucede está en el servidor.

Figura II. 6. RichFaces

Richfaces se describe en varias características, tales como:

- Richfaces proporciona un editor de texto enriquecido.
- Permite agregar capacidades de predicción de texto a un campo de entrada, se le puede indicar el mínimo de caracteres requeridos para mostrar la predicción del texto y la lista sugerida puede contener imágenes, no solo texto.

- Proporciona un componente de ventana modal al cual se pueden asignar acciones a distintos eventos: onbeforeshow, onhide, onmove, onresiza, onmaskmouseout. El estado del contenido de la ventana modal después de hacer submit puede ser guardado.
- Permite seleccionar el modo de la petición de datos de varios componentes visuales.
- Una lista de registros puede ser paginada de manera muy sencilla, mostrando página a página los registros mediante un control de botones de avance y retroceso.

Soporta servidores como:

- Apache Tomcat 5.5 y 6.0
- JBoss 4.2.x y 5.1
- Glassfish 2.1.1
- Weblogic Server 10
- IBM WebSphere
- Jetty
- Geronimo
- NetWeaver

Richfaces se integra con diferentes componentes para poder realizar aplicaciones más potentes, como:

- Sun JSF RI
- Apache MyFaces
- Facelets
- JBoss Seam

La arquitectura de RichFaces consiste en lo siguiente, como se muestra en la figura 7:

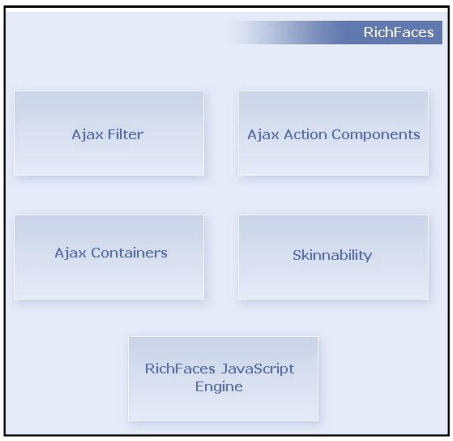


Figura II. 7. Arquitectura Richfaces

Ajax filtro. *Ajax Filter.* Con el fin de obtener todos los beneficios de RichFaces, un desarrollador debe registrar un filtro en el archivo web.xml de la aplicación. El filtro reconoce varios tipos de demanda.

Componentes Ajax de acción. *Ajax Action Components.* AjaxCommandButton, AjaxCommandLink, AjaxPoll y AjaxSupport y otros componentes de acción se pueden utilizar para enviar peticiones Ajax desde el lado del cliente.

Contenedores Ajax. *AjaxContainer* es una interfaz que describe un área de una página JSF que debe ser decodificado en una petición Ajax. AjaxViewRoot y AjaxRegion son implementaciones de esta interfaz.

Skinnability. Personaliza la apariencia de las aplicaciones JSF.

Motor de JavaScript. El motor de JavaScript RichFaces se ejecuta en el lado del cliente. Se actualiza áreas diferentes en una página JSF basado en la información de la respuesta Ajax. El motor de JavaScript proporciona una API para que los desarrolladores no necesiten crear su propia funcionalidad de JavaScript.

RichFaces proporciona dos conjuntos de bibliotecas de componentes:

Core Ajax: la biblioteca principal contiene componentes que son útiles para ajaxizar páginas JSF y componentes estándar de JSF, invocando peticiones Ajax. Además, proporciona un componente de generación de recursos binarios sobre la marcha, por ejemplo, el código generado por las imágenes, archivos PDF, archivos CSV, etc.

La interfaz de usuario: La biblioteca RichFaces interfaz de usuario es un conjunto de componentes avanzados de JSF que Ajax utiliza para agregar características de la interfaz de usuario ricas a sus aplicaciones.

Ajax4JSF y RichFaces se integran totalmente en la arquitectura de JSF y hereda las funcionalidades de sus etiquetas dotándolas con tecnología Ajax de forma limpia y sin añadir código Javascript. En definitiva Ajax4jsf y Richfaces permiten dotar a una aplicación JSF de contenido mucho más profesionales con muy poco esfuerzo, dando como resultado un conjunto de etiquetas listadas a continuación en la Tabla II:

Tabla II. II. Componentes Ajax4JSF y RichFaces

TIPO	ETIQUETA	DESCRIPCIÓN
Ajax4jsf	< aj4:commandButton >	Botón de envío de formulario en el que se puede indicar que únicamente actualice ciertos componentes evitando la recarga de todo el formulario.
	< aj4:commandLink >	Comportamiento similar a < aj4:commandButton > pero en un link
	< aj4:htmlCommandLink >	Muy parecida a la etiqueta anterior con pequeñas diferencias en la generación de links y cuando se utilizan etiquetas < f:param >.
	< aj4:region >	Determina un área a decodificar en el servidor después de la petición Ajax.
	< aj4:status >	Muestra el estado de la petición Ajax: procesando petición y petición terminada.
	< aj4:form >	Similar al < h:form > con la diferencia de que se puede enviar previamente el contenido al contenedor Ajax
	< aj4:actionparam >	Combina la funcionalidad de la etiqueta < f:param > y < f:actionListener >.
	< aj4:outputPanel >	Se utiliza para agrupar componentes para aplicarles similares propiedades.
	< aj4:ajaxListener >	Similar a la propiedad actionListener o valueChangeListener pero con la diferencia de que la petición se hace al contenedor Ajax.
	< aj4:jsFunction >	Se utiliza para pasarle un valor automáticamente a una función JavaScript tras recibirlo del servidor
	< aj4:loadScript >	Inserta en la página las funciones JavaScript contenidas en un archivo *.js
	< aj4:loadStyle >	Igual que la etiqueta anterior pero para una hoja de estilos *.css
	< aj4:log >	Carga en la página una consola que muestra las trazas de los logs que devuelve el contenedor Ajax
	< aj4:include >	Se utiliza para incluir en la página el contenido de otra de acuerdo a la definición que se haga en las reglas de navegación del faces-config.
	< aj4:repeat >	Etiqueta para iterar sobre una colección y mostrar todos sus campos
	< aj4:keepAlive >	Permite mantener un bean en un estado determinado durante peticiones.
	< aj4:mediaOutput >	Componente que permite mostrar contenido multimedia.
	<a4j:push>	Realizar periódicamente petición AJAX al servidor, para simular los datos de inserción.
RichFaces	< rich:calendar >	Componente para crear elementos de calendario.
	< rich:comboBox >	Proporciona combobox editable.
	< rich:contextMenu >	Se utiliza para la creación de "multileveled context menus".

	< rich:dataFilterSlider >	Se utiliza para crear un filtrar de los datos de una tabla.
	< rich:datascroller >	Diseñado para proporcionar la funcionalidad de los cuadros de desplazamiento utilizando solicitudes Ajax.
	< rich:columns >	Permite crear columnas dinámicas.
	< rich:columnGroup >	Permite combinar las columnas en una fila para organizar.
	< rich:dataGrid >	Permite ver los datos como una rejilla que nos deja elegir los datos.
	< rich:dataList >	Permite prestar los datos de un modo lista.
	< rich:dataOrderedList >	Se usa para ordenar las listas de prestación que permite elegir los datos.
	< rich:dataTable >	Permite crear tablas de datos.
	< rich:subTable>	Se utiliza para la inserción de sub tablas
	< rich:dropDownMenu >	Se utiliza para crear múltiples menús desplegables.
	< rich:menuGroup >	Se utiliza para definir un ampliable grupo de temas dentro de una lista emergente u otro grupo.
	< rich:menuItem >	Se utiliza para la definición de un único punto dentro de una lista emergente.
	< rich:menuSeparator >	Se utiliza para la definición de un separador horizontal que puede ser colocado entre los grupos o los temas del programa
	< rich:inplaceSelect >	Muy parecido al anterior. Se utiliza para seleccionar algo así como un dropDown.
	< rich:listShuttle >	Se utiliza para mover los temas elegidos de una lista a otra con su facultativo reordenamiento.
	< rich:message >	Se utiliza para hacer un solo mensaje a un componente específico

Fuente: http://showcase.richfaces.org/

El funcionamiento es sencillo, mediante sus propias etiquetas se generan eventos que envían peticiones al contenedor Ajax.

Estos eventos se pueden ejecutar por pulsar un botón, un enlace, una región específica de la pantalla, un cambio de estado de un componente, etc. Esto significa que el desarrollador no debe preocuparse de crear el código JavaScript y el objeto XMLHttpRequest para que envíe la petición al servidor ya que el framework lo hará.

La primera versión comercial fue lanzada en marzo de 2006 con el nombre de Exadel VCP. En el mismo año, se dividió en dos proyectos Ajax4Jsf (Open Source) y RichFaces (Versión Comercial). *Ajax4jsf* proporcionan el marco básico y los componentes Ajax para ajaxizar los componentes JSF en una página. *RichFaces* fue un componente JSF comercial para las bibliotecas Ajax.

En marzo de 2007, Exadel y JBoss anunciaron una alianza para abrir el código fuente de RichFaces y los dos proyectos se fusionaron en un único proyecto de fuente abierta llamado simplemente RichFaces. La versión 3.3.2, publicada en octubre del 2009 incorpora un conjunto considerable de mejoras y nuevas funcionalidades como las etiquetas para layout, disposición de los elementos de la página.

Esta librería de Componentes posee varias ventajas, entre las cuales se mencionan:

- Aprovecha al máximo los beneficios de JSF framework incluyendo, la validación y conversión de instalaciones, junto con la gestión estática y dinámica los recursos.
- Facilita utilización de AJAX creando dos librerías: a4j y rich.
- Kit de desarrollo de componentes (CDK)
- Dinámica de manejo de recursos
- Amplio soporte multi-navegador

- Ofrece crear una interfaz de usuario enriquecida y de moderna apariencia.

Así mismo ésta librería de componentes se ve en desventaja por:

- Usando Ajax4JSF se debe indicar qué parte de la pantalla tiene que repintarse. Lo que implica tener más control sobre los eventos que se producen en la interfaz de usuario.

2.3.2. ICEFACES

Permite desarrollar Aplicaciones RIA de una manera fácil y rápida, utilizando el lenguaje Java. Está basado en una serie de estándares que permiten continuar trabajando con la clásica forma de desarrollo con Java. De esa manera, se puede trabajar con herramientas existentes como NetBeans, Eclipse; y Servidores de Aplicaciones como Sun GlassFish, Apache Tomcat, IBM WebSphere, JBoss, entre otros.

ICEFaces, figura 8, se encarga de enviar sólo la información necesaria entre cliente y servidor. Es decir, ya no se envían los formularios a la antigua usanza, en un POST de HTTP, sino que sólo se envían los cambios que ha hecho el usuario del cliente al servidor, y los cambios en la pantalla del servidor al cliente.

Figura II. 8. IceFaces

La arquitectura de ésta librería de componentes se muestra en la siguiente figura:

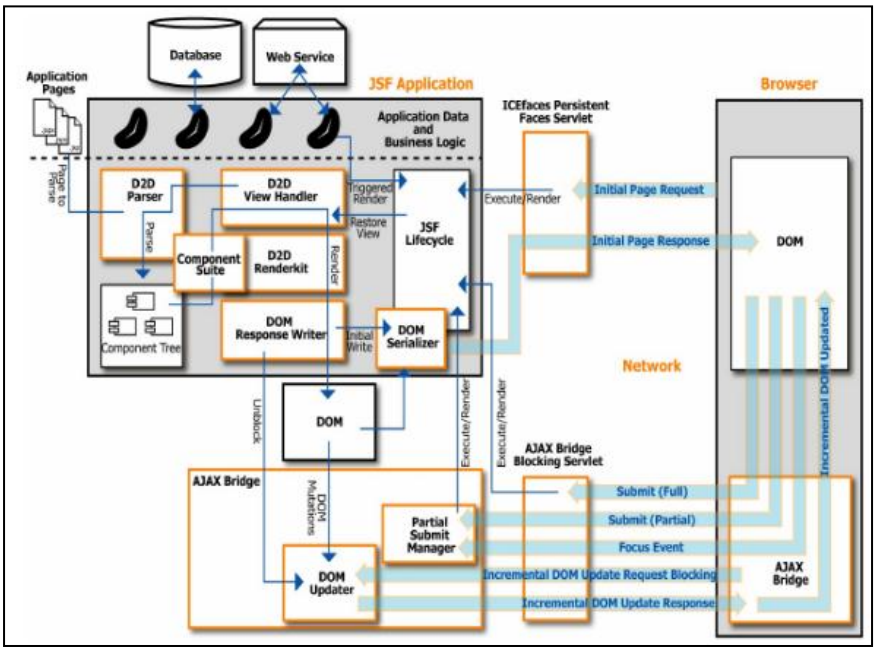


Figura II. 9. Arquitectura JSF con IceFaces

Los principales elementos de la arquitectura ICEfaces incluyen:

- Persistent Faces Servlet: Cuando se realiza una petición de la página inicial en la aplicación, este servlet se hace responsable de la ejecución del ciclo de vida JSF para petición asociada.
- Blocking Servlet: Se encarga de la gestión de todas las peticiones de bloqueo y no-bloqueo después de las primeras páginas.
- D2D ViewHandler: Se encarga de establecer el Direct-to-DOM, incluyendo la inicialización de la 'DOMRespuesta Writer'.
- Parseador D2D: Responsable del montaje de un componente de documentos JSP.
- DOM Response Writer: Se encarga de la escritura en el DOM. También inicia la serialización DOM para la primera prestación, y desbloquea el DOM Updater para actualizaciones incrementales.

- DOM Serializer: Responsable de la serialización del DOM de la página inicial.
- DOM Updater: Se encarga de conjuntar las de las 'DOM mutations' en una única actualización DOM.
- Component Suite: Ofrece un conjunto de componentes 'rich JSF' con influencia AJAX.
- Client-side AJAX Bridge: Responsable de la actualización DOM en curso generada por la solicitud y la respuesta del proceso. También es el encargado de centrar la gestión y de presentar el proceso.

Las aplicaciones desarrolladas en ICEfaces no necesitan plugins de navegador o applets para ser vistas. Estas aplicaciones están basadas en JavaServer Faces, así que permite el desarrollo de aplicaciones Java EE con la posibilidad de utilizar de forma fácil desarrollos basados en JavaScript.

A continuación, en la tabla III se describen algunas de las características de la librería de componentes IceFaces:

Tabla II. III. Características IceFaces

Seguridad	Componentes AJAX	Móviles	Interoperabilidad	Open Source
Compatible con SSL Previene de Cross-Site Scripting (XSS) Previene de ataques de inyección de código malicioso. Protege de ataques de inyección SQL. Previene la minería de datos no autorizados (datamining). Expone la lógica de aplicación o no de datos de usuario	Más de 55 componentes. Integración de efectos de script.aculo.us. Customizables controles con CSS Mecanismo de Drag & Drop	Aplicaciones soportadas en móviles. Minimización de utilización de recursos por lado del cliente. Fácil interacción del usuario con la interfaz.	Compatible con gran variedad de Servidores de Aplicaciones, como Galssfish, Apache Tomcat, JBoss, entre otros. Soportado por diferentes IDEs de desarrollo como Netbeans, Eclipse y MyEclipse Workbench.	Más de 55 auto-componentes Ajax ICEfaces, y más con cada nueva versión. La integración con script.aculo.us efectos JavaScript. Temas a través de CSS. La integración con la biblioteca de componentes Tomahawk.

Fuente: IceFaces wiki http://wiki.icesoft.org/display/ICE/

ICEfaces es una solución de licencia no privativa, que brinda alrededor de 70 componentes, entre ellos, los listados en la Tabla IV:

Tabla II. IV. Componentes IceFaces

ETIQUETA	DESCRIPCIÓN
Checkbox	Casilla de verificación
Column	Única columna de datos en un componente UIData
ColumnGroup	Se usa para crear múltiples encabezados o pies de página de un dataTable con colspan y rowspan
Columns	Sirve para crear varias columnas en una tabla
CommandButton	Crear un botón
CommandLink	Actúa como un botón de envío cuando se hace clic
commandSortHeader	Permite cambiar el orden de los datos de la tabla
DataExporter	Exporta el contenido de datos de una tabla en una variedad de formatos
DataPaginator	Se utiliza para representar controles de navegación de páginas de una tabla
DataTable	Tabla HTML ligada al modelo de datos subyacente.
Effect	Añadir efectos a un componente principal
Form	Creación de formulario
gMap	Google map
gMapControl	Añade controles al Google map
GMapDirection	Dirección de un punto A a un punto B.
GMapGeoXml	Compatible con el formato KML y GeoRSS de datos para la visualización de información geográfica
GMapLatLng	Contenedor para GLatLng Google Map de la API
GMapLatLngs	Lista de componentes gMapLatLngs
GMapMarker	Usa la API GMarker para mostrar un punto en el mapa.
GraphicImage	Elemento de imagen
HeaderRow	Cabecera de fila para una tabla
InputHidden	Elemento de entrada de tipo oculto
InputRichText	Componente JSF de texto enriquecido
InputSecret	Elemento de entrada de tipo clave o password.
InputText	Elemento de entrada de tipo texto
InputTextarea	Elemento área de texto
JsEventListener	Se usa para capturar eventos JavaScript y opcionalmente se envía una notificación al servidor y disparar un evento de acción.
LoadBundle	Permite cambiar a los mensajes de forma dinámica
MenuBar	Proporciona un sistema de menú robusto
MenuItem	Contenido por un menuBar
MenuItems	Jerarquía dinámica de los elementos de menú
menuItemSeparator	Separa los elementos de menú
MenuPopup	Proporciona menús desplegables anidadas
Message	Único mensaje para un componente específico
Messages	Todos los mensajes para un componente específico
OutputChart	Gráficos de varios tipos
outputConnectionStatus	Muestra información acerca del estado de conexión de red.

OutputFormat	Hacer texto parametrizado
OutputLabel	Elemento de tipo etiqueta
OutputLink	Elemento de anclaje o link
OutputMedia	Objeto multimedia
OutputProgress	Informar sobre el progreso a los usuarios en caso de una larga lista de tareas en el servidor
OutputResource	Exponer los recursos para abrir o descargar
OutputStyle	Enlace a hojas de estilo CSS
OutputText	Elemento para mostrar el texto
PanelBorder	Contenedor de componentes con cinco regiones: norte, sur, este, oeste y centro
PanelCollapsible	Dar click en la cabecera para mostrar u ocultar contenido
panelConfirmation	Diálogo de confirmación emergente
PanelDivider	Panel divisible
PanelGrid	Elemento tabla
PanelGroup	Contenedor para un grupo de componentes secundarios
PanelLayout	Colocación de los componentes en la posición absoluta o relativa.
panelPopup	Ventanas emergentes modales o no modales
PanelPositioned	Serie de componentes-hijo repetitivos que pueden ser arrastrados y reposicionado
PanelSeries	Genera dinámicamente una serie de componentes-hijo repetitivos dentro de un panel
PanelStack	Múltiples grupos de panel
PanelTab	Una pestaña dentro de un conjunto de fichas o separadores
PanelTabSet	Un conjunto de pestañas con una tabla activa en un momento
PanelTooltip	Información sobre herramientas emergentes
Portlet	Contenedor para un portal
Radio	Botón de selección para un componente selectOneRadio
Repeat	Mecanismo para generar dinámicamente una serie de repeticiones de componentes
RowSelector	permite la selección de filas únicas y múltiples de una tabla de datos
selectBooleanCheckbox	Elemento de entrada HTML de tipo "checkbox"
SelectInputDate	Campo de entrada, calendario en línea o calendario emergente para introducir la fecha y la hora
SelectInputText	Componente de entrada de texto, autocompletar
selectManyCheckbox	Una lista de casillas de verificación o checkboxes.
selectManyListbox	Cuadro de lista permite selecciones múltiples
SelectManyMenu	Cuadro de lista, permite selecciones múltiples
SelectOneListbox	Cuadro de lista con una única selección.
SelectOneMenu	Cuadro de lista, permite la selección individual
SelectOneRadio	Conjunto de botones tipo radio.
SetEventPhase	Especifica la fase de que ciertos eventos se transmitirán
tabChangeListener	Establece la clase de escucha de cambio de fichas en tabset
Tree	Muestra datos jerárquicos como un árbol de ramas y nodos
TreeNode	Un nodo en un árbol

Fuente: IceFaces wiki http://wiki.icesoft.org/display/ICE/ICE+Components+Reference

COMPONENTES RICOS

IceFaces Component Suit. Incluye implementaciones mejoradas de los componentes JSF estándar y otros componentes personalizados que aprovechan plenamente las IceFaces Direct-to-DOM tecnología de renderizado y ofrecer más características específicas de IceFaces, tales como automatizado parcial, actualizaciones incrementales de páginas y componentes fácilmente configurables look-and-feel. Los componentes del ICEFaces son un conjunto completo de componentes

IceFaces Enterprise Components. Son los componentes exclusivos de ICEfaces 2.0, que derivan de la versión de ICEfaces 1.8. Sin embargo, sólo se pueden usar con la última versión, pues fueron adaptados a la nueva arquitectura de JSF 2.0. Estos componentes disminuyen el tiempo de implementación.

IceFaces Advanced Components. Son los componentes que se basan en **Advanced Component Environment (ACE),** que básicamente son controles automatizados y optimizados, que tratan de cumplir con la funcionalidad de tareas de comunes y que el desarrollador utilice componentes JavaScript, sin la necesidad de aprender o usar directamente JavaScript, algunos ejemplos en la figura 10.

Figura II. 10. Componentes Avanzados IceFaces

Hace uso de potentes componentes con JavaScript, sin tener que desarrollar directamente con JavaScript. Además de poseer soporte de skins para componentes, con capacidades de accesibilidad W3C ARIA, incluyendo navegación por teclado.

Icefaces ha ido evolucionando, presentando varias versiones de la misma:

- ICEfaces Technology Preview-Junio 2005
- ICEfaces 0.1.0 Early Access Release-Agosto 2005
- ICEfaces 0.2.0 Alpha-Octobre 2005
- ICEfaces 0.3.0 Alpha-Enero 2006
- ICEfaces 1.0 Beta-Abril2006
- ICEfaces 1.0 CommunityEdition-Mayo 2006
- ICEfaces 1.5 OpenSource-Novembre 2006
- ICEfaces 1.6 Julio 2007
- ICEfaces 1.7 Abril 2008
- ICEfaces 1.8 Febrero 2009
- ICEfaces 2.0.0 Diciembre 2010
- ICEfaces 3.0.0 Enero 2012

En cuanto a las ventajas que brinda Icefaces se describen las siguientes:

- Muy fácil de aprender porque es muy parecido a JSF y el uso de Ajax es automático, totalmente transparente para el desarrollador.
- Experiencia de usuario enriquecedora.
- Está bajo licencia de código abierto.

- Hay gran cantidad de plugins desarrollados para que ICEfaces sea integrado con variedad de IDEs Java.
- Carga de páginas incremental con edición de secciones y sin recargas de página completas.
- En aplicaciones de tiempo real, las recargas de páginas son asíncronas.

Pero muestra varias desventajas tales como:

- La conexión asíncrona entre el navegador y el servidor se pierde debido a una interrupción de la conexión, la recarga de la página restablece el estado previo de la aplicación.
- Se limita a trabajar con sólo aquellos componentes para los queda soporte.
- Documentación pobre.

2.3.3. Jquery4jsf

JQuery4JSF, figura 11, es una biblioteca de código abierto para Java Server Faces. El propósito de esta biblioteca es poner a disposición el desarrollo de un conjunto de componentes que contienen tecnología RIA, y permitir que el programador pueda realizar rápidamente sus aplicaciones.

Figura II. 11. JQUERY4JSF

Esta biblioteca se basa en el uso de otro marco de código abierto que es JQuery, el mismo que está diseñado para cambiar la forma de programar en JavaScript. Las principales funciones de JavaScript han sido reemplazadas por estructuras que faciliten la vida de un

codificador. Así nació el propósito de crear una biblioteca Jsf que facilite la integración con jQuery y que pondría al desarrollador un número de componentes más personalizado.

Para iniciar JQuery4JSF da a conocer una serie de widgets que son sólo la biblioteca principal de la interfaz de usuario jQuery y un número de otros componentes que tienen como base el plugin de jQuery más famoso.

Varios de los componentes que usa esta librería son:

- Autocompletar
- TextEditor
- DatePicker
- Tabs
- InputMaked
- TreeView
- Acordeón
- ToolTip

2.3.4. Primefaces

Primefaces, figura 12, librería de componentes Java para la capa de la vista en el modelo MVC, ligero y Open Source. Es realmente impecable, es no intrusivo por lo que se puede utilizar con otros frameworks Java, basado en la librería JQuery lo que hace que los componentes sean realmente rápidos.

Figura II. 12. PrimeFaces

Fue el primer framework JSF en soportar la especificación JSF 2.0.

PrimeFaces es una librería de componentes visuales Open Source desarrollada y mantenida por Prime Technology, sus principales características son:

- Soporte nativo de Ajax, incluyendo Push/Comet.
- Kit para crear aplicaciones web para móviles.
- Es compatible con otras librerías de componentes, como JBoss RichFaces.
- Uso de javascript no intrusivo, aparece dentro de un bloque <script>.
- Es un proyecto open source, activo y bastante estable entre versiones.

PrimeFaces 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.

Conjunto de componentes Ajax

- AjaxEngine, corre bajo las normas de API JSF 2.0.
- AJAX Poll, Componente de sondeo se utiliza para realizar periódicamente peticiones ajax y ejecutar comandos.
- Ajax Status, es un indicador global para informar a los usuarios acerca de las interacciones AJAX.

En la Tabla V se muestra un listado de varios componentes de esta librería de componentes Primefaces.

Tabla II. V. Componentes Primefaces

Tipo	Componente	Descripción
Ajax Core	Basics	Muestra las características de actualización del Ajax básico.
	Partial Processing	Permite elegir los componentes para procesar mediante el atributo process. Útil para hacer validaciones parciales, actualización de modelos, invocar la aplicación y mucho más. Palabras clave como: @this, @form, @all, @none, @parent hacen que sea aún más fácil elegir lo que debe procesar. sólo para ajax
	Partial Submit	Reduce el tráfico de red mediante la adición de los componentes sólo parcialmente procesados para el cargo petición ajax. Para las páginas grandes con muchos componentes de entrada, partialSubmit es muy útil ya que da lugar a las peticiones más ligeras.
	Validations	Se ejecutan en el servidor y la página se actualiza con ajax.
	Selectors	La potencia total de Selector API se puede utilizar con o sin el componente referenciamiento regular.
	Polling	Hace llamadas Ajax en un intervalo especificado.
	RemoteCommand	Permite ejecutar métodos de respaldo de beans y hacer actualizaciones parciales provocadas por secuencias de comandos personalizadas del lado del cliente.
	AjaxStatus	Es un indicador global para informar a los usuarios acerca de las interacciones Ajax.
	Events	Habilita el comportamiento Ajax en cualquiera de los componentes JSF.
	Selects	Tiene que ver con selectOneMenu PrimeFaces que es un estándar con capacidades de edición, efectos, filtrado y visualización de contenido personalizado.
	Listeners	Componente p:ajax ejecuta un listener en un bean JSF.
	Counter	Una variable entera que cuenta.
Input	AutoComplete	Autocompletar en un campo de entrada.
	BooleanButton	Se utiliza para proporcionar selección binaria con una interfaz de usuario de botón en lugar de una casilla de verificación.
	BoolCheckbox	Extiende SelectBooleanCheckbox estándar con capacidades de desglosar.
	Calendar	Es un componente de selector de fecha de gran alcance que ofrece diferentes modos de visualización
	CheckboxMenu	Es un componente de entrada de selección múltiple basado en casillas de verificación en un menú.
	ColorPicker	Selección de colores en lugar de escribir el código hexadecimal.
	Editor	Es un componente de entrada con ricas características de edición de texto.
	Inplace	Para la edición in-situ, "guardar" y "cancelar".
	InputMask	Componente de entrada de para ser formateado de manera específica.
	InputText	Extiende el InputText estándar con capacidades de desglosar.
	InputTextarea	Extiende el estándar InputTextarea con autoResize, barra y características de tematización.

	Keyboard	Muestra el teclado en pantalla, según lo requerido puede mostrar el teclado completo, sólo alfabético, sólo numérico, tipo password, etc.
	ManyButton	Es un componente de entrada para seleccionar las opciones con los botones en vez de casillas de verificación.
	ManyMenu	Extiende el SelecManyMenu con personalización.
	ManyCheckbox	Extiende el SelectManyCheckbox con capacidades de desglose.
	OneButton	Componente de entrada para seleccionar las opciones con los botones regulares en lugar de botones de radio.
	OneMenu	Extiende el SelectOneMenu estándar con capacidades de desglose, edición, efectos y visualización de contenido personalizado.
	OneListbox	Extiende el SelectOneListBox estándar con personalización.
	OneRadio	Extiende el SelectOneRadio estándar con capacidades de aplicación de aspectos y características de diseño personalizado.
	Password	Versión extendida del componente inputSecret integrando temas e indicador de intensidad o fuerza de la clave.
	Rating	Componente que provee una entrada de estrellas basada en una calsificación.
	Spinner	Usado para proporcionar entradas de botones con incrementos o decrementos en una entrada de texto o inputText.
	Slider	Usado para mostrar entradas de varias formas con una barra de desplazamiento vertical u horizontal.
Button	Button	Es una extensión de la etiqueta estándar h:button, recibe peticiones dirigidas a las direcciones URL.
	CommandButton	Extensión de h:commandLink con Ajax, procesamiento parcial y características de confirmación.
	CommandLink	Extensión de h:commandButton con Ajax, procesamiento parcial y capacidad de desglose.
	SplitButton	Muestra un comando por defecto.
Data	Carousel	Componente multiuso para mostrar un conjunto de datos o contenido general. Tiene Simple Carousel, Item Selection, Effects, Tab Slider, SlideShow
	DataExporter	Sirve para exportar tabla de datos a varios formatos como Excel, pdf, csv y xml.
	DataList	Presenta una colección de datos en una lista con diseño y varios tipos de visualización, Características como: Unordered DataList, Ordered DataList, Definition DataList, Ajax Pagination.
	DataGrid	Muestra datos en una cuadrícula diseñada. Ajax Pagination es una característica integrada y la interfaz de usuario es totalmente personalizable
	DataTable	Es un componente iteración datos con paginación ajax, ordenar, filtrar varias soluciones de selección de fila, encabezados anidados, filas expansibles, edición en la celda, desplazamiento..
	PickList	Es un componente de lista de entrada dual con arrastrar y soltar basada con reordenamiento, efectos de transición, soporte para temas y más.
	OrderList	Se utiliza para clasificar una colección a través de arrastrar y soltar reordenamiento basado, efectos de transición, el apoyo

		pojo, soporte para temas y más.
	Sheet	Es un componente parecido a una tabla Excel, con manipulación de datos, ordenamiento ajax, columnas redimensionables, desplazamiento horizontal o vertical, teclas de navegación, etc.
	Ring	Es un componente de visualización de datos con una animación circular.
	Tree	Se utiliza para mostrar los datos de cualquier jerarquía o propósito de navegación. Con opciones de estilo, ajax expandir, contraer y seleccionar los eventos, una función de selección basada en casilla de verificación
	TreeTable	Usado para mostrar jerarquías de datos en un formato tabular
Panel	Accordion	Es un componente contenedor con los paneles verticalmente apilados.
	Dashboard	Basado en la reordenación.
	Fieldset	Componente de agrupación
	Layout	Diseño de borde que puede ser aplicado a una página completa o sólo partes de ella, puede expandir, contraer, redimensionar, cerrar.
	NotificationBar	Muestra un panel fijo de usos múltiples ubicado para notificación ya sea en la parte superior o inferior de la página. Cualquier grupo de componentes JSF puede colocarse dentro de la barra de notificación.
	OutputPanel	Es un elemento contenedor con varios casos de uso.
	Panel	Es un componente de agrupación genérica que también soporta comunicación alternativa, el cierre y el menú de opciones.
	PanelGrid	Es una extensión del panelGrid estándar con la integración de temas y soporte colspan-rowspan.
	ScrollPanel	Se utiliza para mostrar cantidades grandes de contenido con barras de desplazamiento.
	TabView	Es un componente poderoso para panel de pestañas con pestañas, carga de contenido dinámico con ajax, orientaciones diferentes, la creación / eliminación de fichas mediante programación y mucho más.
	Toolbar	Componente para agrupación de botones y otros contenidos.
	Wizard	Asistente de flujo es secuencial por defecto y esto puede ser logrado con el programa opcional flowListeners ajax, simplemente resultado de una flowListener define el siguiente paso para mostrar. Paso actual se procesa parcialmente y el paso siguiente se muestra si pasa la validación.
Overlay	ConfirmDialog	Cuadros de diálogos de confirmación.
	Dialog	Es un componente contenedor que puede colocar otros elementos en la página. Diálog tiene varias opciones de personalización, tales como modal, cambiar el tamaño, anchura, altura, posición. Cuadro de diálogo.
	LightBox	Es un componente de superposición modal para mostrar imágenes, contenido en línea e iframes.
	OverlayPanel	Es un componente contenedor genérico que puede sobreponer otros componentes en la página. Puede presentarse en un panel básico, dinámico o a modo de imagen.

	Tooltip	Muestra mensajes de ayuda.
Menu	BreadCrumb	Proporciona información contextual sobre la jerarquía de página.
	ContextMenu	Es un menú de superposición que se muestra al dar click en el botón derecho del ratón que se puede conectar a cualquier otro componente JSF.
	Dock	Acoplar menús en enlaces.
	MegaMenu	Muestra los submenús de elementos raíz juntos.
	Menu	Es personalizable, componente de comando y navegación que soporta posicionamiento dinámico y estático.
	Menubar	Aporta las barras de menú de aplicaciones de escritorio para JSF
	MenuButton	Agrupa varios comandos en un menú emergente.
	PanelMenu	Componente híbrido de árbol-acordeon usado para navegación y acción
	SlideMenu	Muestra submenús con animación slide.
	TabMenu	Componente menú que muestra ítems en pestañas
	TieredMenu	Muestra submenús anidados
Charts	Area	Activa opciones de apilado y relleno de lineChart
	Bar, Bubble, Donut, Line, Pie, MeterGauge, OHLC, Animate Export, Interactive, Live, Update, Static, Zoom	Componentes que permita mostrar gráficos estadísticos en forma de barras, líneas, etc.
Message	Growl Messages	Muestra avisos de mensajes
Multimedia	Compare	Provee interfaz gráfica para comparar imágenes similares.
	Cropper	Usado para hacer recortes de imágenes.
	DynaImage	Presenta imágenes creadas en tiempo de ejecución o de la base de datos
	Galleria	Componente para galería de imágenes.
	Media	Incrusta audio y video.
	PhotoCam	Captura fotos desde la cámara del computador.
	Switch	Soporta 25 efectos
File	Auto	Carga de archivo automáticamente llama ventana de archivos
	Basic	Examina rutas para cargar archivos.
	DragDrop	Seleccionar un archivo, cargarlo o cancelar desde un panel con pestañas.
	Download	Descarga de archivos
	Multiple	Carga de varios archivos.
	Single	Carga de archivo único.
DragDrop	Draggable	Permite el movimiento de componentes horizontal, vertical, área definida, con efectos y otras características.
	DataGrid	Integración con componente grid
	DataTable	Integración con componente tabla.

Misc	Captcha	Componente para crear captcha o test controlado.
	Collector	Componente para manejar colecciones sin código java.
	DefaultCommand	Controla que componente inicia un formulario
	Effects	Basado en jQuery, varios efectos de ventanas.
	ProgressBar	Barra de estado de proceso
	Resizable	Permite cambiar el tamaño de cualquier componente.
	Separator	Muestra una línea horizontal cuando se usa como un componente individual, también para separador de menús y barra de herramientas.

Fuente: http://www.primefaces.org/showcase-labs/ui/home.jsf

En la Tabla VI se muestra un listado de los cambios o versiones que ha tenido Primefaces:

Tabla II. VI. Versiones PrimeFaces

VERSIÓN	FECHA
1.1/	26-Julio-2010
2.1/	26-Julio-2010
2.2/	06-Febrero-2011
2.2.1/	15-Febrero-2011
3.0.M1/	11-Abril-2011
3.0.M2/	04-Julio-2011
3.0.M3/	29-Agosto-2011
3.0.M4-SNAPSHOT/	04-Octubre-2011
3.0.M4 /	31-Octubre-2011
3.0.RC1-SNAPSHOT /	31-Octubre-2011
3.0-SNAPSHOT/	11-Diciembre-2011
3.0.RC1 /	11-Diciembre-2011
3.0.RC2-SNAPSHOT /	14-Diciembre-2011
3.0.RC2/	18-Diciembre-2011
3.0.1-SNAPSHOT/	11-Enero-2012
3.0.1/	15-Enero-2012
3.0/	02-Enero-2012
3.1-SNAPSHOT/	02-Enero-2012
3.1.RC1/	29-Enero-2012
3.1/	05-Febrero-2012
3.2-SNAPSHOT/	05-Febrero-2012

3.1.1/	14-Febrero-2012
3.2.RC1/	05-Marzo-2012
3.2/	11-Marzo-2012
3.3-SNAPSHOT/	11-Marzo-2012

Fuente:http://aplicacioneslibres.bligoo.ec/media/users/19/951967/files/213572/PrimefacesJonathanGuerrero.pdf

Primefaces brinda las siguientes ventajas:

- En cuanto a la experiencia de los usuarios finales los componentes de PrimeFaces son amigables al usuario además que cuentan con un diseño innovador.
- Tiene una gran cantidad de componentes libres, las aplicaciones se sienten fluidas y el uso de Ajax es sencillo.
- Soporte Ajax transparente al desarrollador.
- Usa soporte de jQuery para los efectos visuales.
- Cuenta con más de 100 componentes Open Source, algunos de muy alta calidad.
- Dispone de un kit para crear interfaces web para teléfonos móviles.
- Está integrado con ThemeRoller Framework CSS, de donde se puede elegir el tema a su preferencia.
- Es estable, con gran calidad de componentes, skins.
- Posee una buena documentación.
- Se actualiza muy rápidamente a comparación de otros frameworks.

Entre las falencias de esta librería de componentes se menciona que:

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

- Un componente de una nueva versión de Primefaces por lo general no es compatible al 100% con una aplicación desarrollada con la versión previa del componente. Lo cual conlleva a dedicarle demasiado esfuerzo en poder adecuar una aplicación existente para que pueda utilizar la nueva versión de los componentes PrimeFaces.

2.3.5. Openfaces

OpenFaces, figura 13, es una biblioteca de código abierto de AJAX. OpenFaces se basa en el conjunto de componentes JSF antes conocido como QuipuKit. Contiene código base completamente revisada de QuipuKit e introduce muchos nuevos componentes y características.

Figura II. 13. OpenFaces

La biblioteca OpenFaces incluye un amplio conjunto de componentes JSF para construir ricas interfaces de usuario Web y un marco de validación que cambia la tradicional lógica de validación JSF para el lado del cliente.

Entre los componentes de OpenFaces, se presentan los siguientes de la tabla VII:

Tabla II. VII. Componentes OpenFaces

COMPONENTE	DESCRIPCIÓN
Ajax Framework	Componente no visual que se puede conectar a cualquier otro componente JSF para recargar otros componentes
Border Layout Panel	Es un contenedor utilizado para grupos de disposición de los componentes por los bordes del contenedor
Calendar	Permite seleccionar una fecha en un calendario de mes y navegar fácilmente entre meses y años
Chart	Se basa en el motor JFreeChart, representa varios conjuntos de datos en forma gráfica, por ejemplo como un pastel, líneas o gráficos de barras.
Command Button	Versión extendida del componente JSF, CommandButton.
Command Link	Versión extendida del componente JSF, CommandLink.
Confirmation	Permite a los usuarios confirmar o rechazar las acciones críticas antes de su ejecución. Se muestra a través de elementos de la página, como un cuadro de diálogo moda
Data Table	Se utiliza para mostrar los datos en formato de tabla

Data Chooser	Permite al usuario introducir una fecha, ya sea escribiéndolo en el campo de texto o seleccionándolo del calendario desplegable. Con apoyo de internacionalización, una fecha seleccionada se puede visualizar en diferentes formatos de fecha y lenguas.
DayTable	Permite la creación, visualización, edición de eventos diarios.
Drop Down Field	Componente de entrada, lista desplegable.
Dynamic Image	Muestra imagen en momento de ejecución.
Hint Label	Se usa para texto que no cabe en el espacio asignado, para mostrar ayudas, cuando se coloca el mouse sobre un elemento.
Input Text	Componente de entrada de texto.
InputTextArea	Componente de entrada de texto multilínea.
LayeredPanel	Es un contenedor que permite cambiar entre componentes o pestañas
PopupMenu	Se usa como menú contextual o menú emergente.
SelectBooleanCheckbox	Casillas de verificación, selección de una sola opción
SelectManyCheckbox	Casillas de verificación, selecciona varias opciones.
SelectOneRadio	Soporta características estándar del componente JSF RadioButton.
Spinner	Campo de entrada de números, campo de texto donde se adjunta el incremento o decremento.
SuggestionField	Campo de entrada de texto con filtrado.
Tab Set	Conjunto de pestañas pero no en un contenedor.
TabbedPanel	Contenedor que permite cambiar de páginas según la creación de pestañas.
Window	Componente que puede ser arrastrado por la página y cambiar de tamaño, además maximizar / restaurar, minimizar y cerrar botones en el título.

Fuente: http://openfaces.org/components/

Todos los componentes OpenFaces soportan los siguientes navegadores:

- Microsoft Internet Explorer
- Mozilla Firefox
- Opera
- Apple Safari
- Google Chrome

Los componentes OpenFaces requieren versiones JDK 1.5 o posteriores.

OpenFaces requiere la versión 2.0 de JSF, para poder ser usada.

OpenFaces ha sido probado para trabajar con los servidores de aplicaciones: Apache Tomcat 6.0 y GlassFish v3.

2.3.6. Apache Myfaces

Apache MyFaces, figura 14, es un proyecto de Apache Software Foundation, para mantener una implementación abierta de JavaServer Faces JSF, por medio del desarrollo de bibliotecas y componentes, proporciona varios subproyectos referente al marco JSF.

Figura II. 14. Apcache MyFaces

Este proyecto provee:

- Una implementación JavaServer Faces.
- Varias bibliotecas de componentes que contienen los widgets de interfaz de usuario para la creación de aplicaciones web con JSF, por ejemplo MyFaces Tomahawk, MyFaces Trinidad, MyFaces Tobago.
- Paquetes de extensión a JavaServer Faces, por ejemplo, MyFaces Orchestra, MyFaces Extensiones Validator, MyFaces Extensiones CDI.
- Módulos de integración con otras tecnologías y estándares, por ejemplo, MyFaces portlet.

El proyecto se divide en:

- ***Core***: es una implementación de JSF 1.1 y 1.2 y los componentes que se han especificado en JSR 127 y 252 respectivamente. Se divide en dos submódulos:
  - MyFaces API implementa todas las clases que están definidas en la especificación.
  - MyFaces Impl ofrece clases invisibles de apoyo cuyo código el usuario no invoca directamente, sino que se necesita un marco de trabajo JSF.

- ***Tomahawk:*** es un conjunto de componentes creados por el equipo de desarrollo de MyFaces y donados a Apache.
- ***Trinidad:*** conjunto de componentes JSF aportados por Oracle, también conocidos como ADF Faces. No es solo una librería de componentes JSF, provee al desarrollador de un moderno framework web que se enfoca en amplitud y una filosofía de diseño de software de mundo cerrado.
- ***Tobago:*** un conjunto de componentes contribuidos a MyFaces por Atanion GmbH. Su objetivo es proporcionar un conjunto bien diseñado de componentes de interfaz de usuario basados en JSF.
- ***Orchestra:*** un framework usado para el control de la persistencia entre sesiones. Utiliza el poderoso mecanismo de configuración de Spring Framework, para administrar el ciclo de vida de los componentes de respaldo (backing beans).

2.3.7. ADF Faces

Oracle Application Development Framework (ADF), proporciona la infraestructura para integrar varias tecnologías empresariales; ayudándole a explotar los patrones de diseño como MVC; y permitiendo que IDEs como Oracle JDeveloper generen código fuente que es compacto y mantenible.

ADF Faces, figura 15, es una serie de componentes JSF hechos por Oracle. Oracle ha donado estos componentes a Apache.

Figura II. 15. ADF Faces

Para una aplicación Web, el marco de trabajo JSF y ADF Faces se configuran utilizando los tres ficheros XML:

- `web.xml`. Empieza con un parámetro de inicialización (DATA_DIR) cuyo valor indica donde se almacenarán los ficheros subidos.
- `faces-config.xml`. Define un bean manejado (adfupload.UploadBean), JSF creará ejemplares de este bean en el ámbito de la request JSP. JSF inicializará las propiedades uploadedFile y override con un valor de null y true.
- `adf-faces-config.xml`. Aquí se especifican los parámetros de inicialización de ADF, como accessibility-mode y look-and-feel, deben especificarse en un fichero sepa.

A continuación en la tabla VIII un listado de varias de las etiquetas que servirán para el desarrollo de una página web:

Tabla II. VIII. Componentes ADF FACES

COMPONENTE	DESCRIPCIÓN
<af:activeCommandToolbarButton>	Crea un botón en una barra de herramientas. Se utiliza normalmente en el interior de un componente <af:toolbar>
<af:activeImage>	Trasmite una imagen especificada por la propiedad de origen y apoya cambiar esta propiedad a través de los datos activos.
<af:activeOutputText>	Soporta texto con estilo y modificación de este texto a través de los datos activos. El texto opcionalmente se puede dejar sin escapar. La conversión hacia y desde los objetos de java no es compatible
<af:breadCrumbs>	Se utiliza en diseños jerárquicos para indicar el camino de vuelta a la página raíz de la jerarquía con los enlaces.
<af:calendar>	Ofrece la posibilidad de ver las actividades en un calendario
<af:carousel>	Muestra una serie hilado de elementos basados en las filas de un modelo de colección.
<af:chooseColor>	Se utiliza en conjunción con un inputcolor para permitir al usuario seleccionar rápidamente un valor de color sin tener que navegar a una ventana secundaria.
<af:chooseDate>	Se utiliza en conjunción con un inputdate para permitir al usuario seleccionar rápidamente un valor de fecha sin tener que navegar a una ventana secundaria.

<af:column>	Componente que se utiliza como un hijo de la componente tabla. El componente de la columna define el encabezado, pie de página y los datos de una sola columna en la tabla.
<af:commandButton>	Crea un botón que, al pulsarlo, generará un evento de acción en el servidor. El botón puede contener texto, una imagen o un texto y una imagen.
<af:commandImageLink>	Crea un vínculo que cuenta con el apoyo de un ícono, y cuando se pulsa, se generará un evento de acción en el servidor.
<af:commandLink>	Crea un enlace que, cuando se pulsa, se generará un evento de acción en el servidor.
<af:commandMenuItem>	Crea una representación elemento de menú que, cuando se pulsa, se generará un evento de acción en el servidor.
<af:commandNavigationItem>	Crea una representación elemento de navegación de un comando de interfaz de usuario
<af:commandToolbarButton>	Crea un botón en una barra de herramientas. Se utiliza normalmente dentro de un componente de barra de herramientas.
<af:contextInfo>	Añade un área que se puede hacer clic para mostrar la información contextual.
<af:declarativeComponent>	Añade un componente declarativo dinámico a la página.
<af:dialog>	Es un elemento de diseño que muestra a sus hijos dentro de una ventana de diálogo y entrega dialogevents cuando las acciones aceptar y cancelar se activan.
<af:document>	Crea cada uno de los elementos fundamentales estándar de una página html: <html>, <body> y <head>.
<af:form>	Crea un elemento <form> html.
<af:goButton>	Crea un botón que se desplaza directamente a otro lugar en vez de la entrega de una acción.
<af:goImageLink>	Un vínculo html con un soporte icono.
<af:group>	Control invisible
<af:icon>	Representa un icono.
<af:image>	Crea una etiqueta de imagen.
<af:inlineFrame>	Se utiliza para crear una etiqueta de marco en línea.
<af:inputColor>	Campo de entrada para los colores
<af:inputComboboxListOfValues>	Se utiliza para seleccionar y devolver datos para los campos en la página principal.
<af:inputDate>	Representa un campo de entrada para las fechas.
<af:inputFile>	Componente que puede ser utilizado para cargar un archivo.
<af:inputText>	Un campo de entrada de texto de control.
<af:media>	Muestra el contenido de medios, tales como audio, vídeo o imagen en un reproductor incrustado en el agente de usuario.
<af:menu>	Componente de menú vertical

<af:menuBar>	Representa un componente de barra de menú
<af:message>	Muestra un mensaje
<af:messages>	Muestra mensajes importantes a nivel de página información de mensajería para el usuario.
<af:navigationPane>	Crea una serie de elementos de navegación que representan un nivel en la jerarquía de navegación.
<af:outputLabel>	Crea una serie de elementos de navegación que representan un nivel en la jerarquía de navegación.
<af:outputText>	Soporta texto con estilo
<af:query>	Se utiliza para mostrar un panel de búsqueda completa.
<af:quickQuery>	Se utiliza para mostrar un panel de búsqueda completa.
<af:richTextEditor>	Crea un widget de entrada para el formato de texto enriquecido.
<af:selectBooleanCheckbox>	Permite al usuario final seleccionar una casilla de verificación.
<af:selectBooleanRadio>	Permite al usuario final seleccionar un botón de opción en un grupo de botones de radio.
<af:selectItem>	Representa un solo elemento que el usuario puede seleccionar de una lista
<af:selectManyCheckbox>	Permite al usuario final seleccionar varios valores de una lista de opciones disponibles.
<af:selectManyChoice>	Permite al usuario seleccionar varios valores de una lista desplegable de elementos.
<af:selectManyListbox>	Crea un componente que permite al usuario seleccionar varios valores de una lista de elementos.
<af:selectManyShuttle>	Proporciona un mecanismo para seleccionar varios valores de una lista de valores que permite al usuario mover elementos entre dos listas
<af:selectOneChoice>	Crea un componente de menú de estilo, que permite al usuario seleccionar un solo valor de una lista de elementos.
<af:selectOneListbox>	Crea un componente que permite al usuario seleccionar un solo valor de una lista de elementos.
<af:selectOneRadio>	Permite al usuario final seleccionar un solo valor de una lista de opciones disponibles
<af:showDetail>	Proporciona un medio de alternar un grupo de componentes está entre ocultar o mostrar.
<af:showDetailHeader>	Proporciona un medio para alternar los contenidos bajo un encabezado entre ser revelada o no divulgada.
<af:showDetailItem>	Representa un solo elemento con contenidos específicos que pueden ser seleccionados por el usuario
<af:subform>	The subform tag represents an independently submittable region of a page. The contents of a subform will only be validated (or otherwise processed) if a component inside of the subform is responsible for submitting the page.
<af:table>	Se utiliza para mostrar datos tabulares. También es compatible con la

	selección simple y múltiple, clasificación y exploración de registros.
<af:toolbar>	Barra de herramientas
<af:toolbox>	Un contenedor para las barras de herramientas y barras de menús
<af:train>	Indica la ubicación de la página actual dentro de un proceso de múltiples pasos

Fuente: http://www.oracle.com/technetwork/developer-tools/

Varias de las mejoras de ADF Faces son:

- ADF Faces ofrece otros muchos componentes UI, incluyendo tablas y árboles, que permitirán al desarrollador enfocarse en su tarea principal cuando construya complejos UIs para la Web.
- ADF Faces tiene en cuenta cosas como el análisis de multipart/form-data, un aspecto y comportamiento consistente, y control de estado del UI, para que pueda construir página Web utilizando componentes UI en lugar de tener que tratar con etiquetas HTML, código Javascript y CCS
- Proporciona APIs para construir componentes personalizados que sean reutilizables entre aplicaciones.

2.3.8. RC Faces

Es una biblioteca JavaServer Faces que proporciona un conjunto de componentes para la construcción de aplicaciones Web modernas. RC Faces usa tecnologías AJAX y una API orientada a objetos JavaScript para facilitar la construcción de interfaces de usuario altamente interactivas.

Este código abierto de la biblioteca ha sido desarrollado por la empresa Vedana desde el verano de 2004.

Ésta librería RC Faces es compatible con los estándares Java Server Faces. Fue diseñada para facilitar desarrollos. Por lo tanto, ofrece numerosos componentes avanzados:

- Puede ordenar, de tamaño variable y listas paginadas de varias columnas, etc. que utiliza la tecnología Web 2.0 Ajax.

- Propiedades de los controles se manejan a través de una API unificada disponible en el servidor y en las partes de los clientes (en el servidor java y javascript en el cliente).
- En la generación actual de imágenes derivadas de una imagen base.
- Recursos Transparente en tiempo de ejecución.

Para hacer un resumen, a continuación en la tabla IX se muestra un listado de varios componentes de RC Faces:

Tabla II. IX. Lista de Componentes de RC Faces

	COMPONENTE	DESCRIPCIÓN
Básico	Box	Es un contenedor. Se puede tener una representación gráfica o no, pero es principalmente utilizada para aplicar un tratamiento colectivo a un conjunto de componentes, por ejemplo mostrar u ocultar un grupo de componente.
	externalBox	Es un contenedor basado en el <iframe> etiqueta HTML estándar.
	FieldSet	Es un contenedor con un título (texto y / o imágenes).
	HelpButton	Es un hipervínculo clásico
	Image	Muestra una imagen
	LineBreak	Es el equivalente HTML .
	Message	Es un marcador de posición para mensajes de error (sólo se muestra uno).
	Messages	Es un marcador de posición para los mensajes de error (varios mensajes se pueden mostrar simultáneamente).
	Ruler	Es el equivalente HTML <HR>
	StyledText	Es un marcador de posición para mostrar "mejorado" de texto. Acepta cualquier etiqueta HTML. que es un componente de texto
	Text	Es un marcador de posición para mostrar el texto
No Visuales	Accelerator	Permite asociar una tecla de aceleración a una acción u otro componente.
	FocusManager	Permite hacer frente a la concentración en la página actual.
	HiddenValue	Permite acceder y almacenar valor en el cliente y en el servidor mientras se mantiene oculto.
	Init	tiene el incluido javascript y css.
	Javascript	incluye clases obligatorias y javascript.
	Service	Permite llamar a los servicios de AJAX desde el cliente.
Entradas	Button	Equivalente al <button> etiqueta HTML estándar.

	CheckButton	Equivalente al <INPUT type="checkbox"> etiqueta HTML.
	DateChooser	Muestra un calendario y ayuda al usuario a elegir una fecha.
	DateEntry	Un componente de entrada de texto especializado.
	ImageButton	Un componente de botón que puede mostrar una imagen.
	ImageCheckButton	Un componente CheckButton con una imagen en lugar de la casilla.
	TextArea	Basado en el <TEXTAREA> etiqueta HTML estándar.
	ImageRadioButton	Un componente RadioButton con una imagen en lugar de la caja redondeada.
Complejos	Calendar	Muestra un calendario.
	ComboGrid	El componente comboGrid tiene las mismas fonctionalities el componente combo, pero la ventana emergente se basa en un DataGrid
	DataGrid	Un componente de red.
	Pager	Muestra información acerca de un conjunto de resultados de componente DataGrid.
	TabbedPane	Proporciona una manera de mostrar más información en una sola página.
	TextEditor	Un contenedor que permite editar texto y tiene una barra de herramientas específica.
	ToolBar	Permite a los desarrolladores añadir fácilmente las barras de herramientas que contienen componentes diferentes.
	Menu	Ofrece una manera de crear menús de escritorio de estilo en las páginas web.

Fuente: http://www.rcfaces.org/comps/comp-complex-info.html

RC Faces puede ser usada en los siguientes servidores:

- Tomcat 5,5 a 6,0
- WebLogic 10,3
- JBoss 5.0
- GlassGish V3

RC Faces fue desarrollado con Apache Tomcat y es compatible en Navegadores como:

- Internet Explorer 6.0 y superior
- Firefox 3.0 y superior 5
- Chrome 10,0 y superior 0
- Safari 5.0

2.3.9. PrettyFaces

PrettyFaces, figura 15, es una extensión de Servlets basado en OpenSource filtro con soporte mejorado para JavaServer Faces 1.1, 1.2 y 2.0. PrettyFaces resuelve el problema de "Descanso URL" elegantemente, incluyendo características tales como: carga de la página acciones, integración con caras navegación, asignación dinámica vista-identificación, análisis del parámetro administrado y libre configuración compatibilidad con otros frameworks web.

Figura II. 16. RC Faces

PrettyFaces es un servlet que intercepta las peticiones realizadas al servidor. PrettyFaces nos permite configuraciones más complejas y potentes. Resuelve varios problemas con elegancia, como por ejemplo: la reescritura de URL personalizado, carga de la página, las acciones de integración sin fisuras con JSF navegación y enlaces, vista dinámico-id asignación, gestionados análisis de parámetros y configuración sin compatibilidad con otros marcos de JSF.

Después de realizar el estudio de las Librerías de Componentes, a continuación en la Tabla X se detalla una breve descripción de cada una de ellas.

Tabla II. X. Librerías de Componentes

Librería de Componente	Descripción
RichFaces (JBoss)	Facilita utilización de AJAX creando dos librerías: a4j y rich
IceFaces (IceSoft)	Se encarga de enviar sólo la información necesaria entre cliente y servidor. Ya no se envían los formularios en un POST de HTTP, sino que sólo se envían los cambios que ha hecho el usuario del cliente al servidor, y los cambios en la pantalla del servidor al cliente. Las aplicaciones desarrolladas en ICEfaces no necesitan plugins de navegador o applets para ser vistas
jQuery4jsf	Basado en jQuery. Componentes que tienen como base el plugin de jQuery
PrimeFaces	Es una librería muy liviana, y muy simple que no necesita dependencias y configuraciones.

	No intrusivo Kit para crear aplicaciones web para móviles Compatible con otras librerías de componentes Integrado con ThemeRoller Framework CSS Primer framework JSF en soportar la especificación JSF 2.0
OpenFaces	Se basa en el conjunto de componentes JSF antes conocido como QuipuKit
MyFaces(Apache)	Apache MyFaces es un proyecto de la Fundación Apache Software, y alberga varios sub proyectos relacionados con la tecnología JavaServer Faces como son: Core, Tomahawk, Trinidad, Tobago, Orchestra.
ADF Faces (Oracle)	Una serie de componentes JSF hechos por Oracle y donados a la Fundación Apache. Proporciona APIs para construir componentes personalizados que sean reutilizables entre aplicaciones.
RC Faces (Oracle)	Orientada a objetos JavaScript para facilitar la construcción de interfaces de usuario altamente interactivas. Recursos Transparente en tiempo de ejecución.
PrettyFaces (OCPSoft)	Es una extensión de Servlets Resuelve varios problemas con elegancia, como por ejemplo: la reescritura de URL personalizado, carga de la página, las acciones de integración sin fisuras con JSF navegación y enlaces

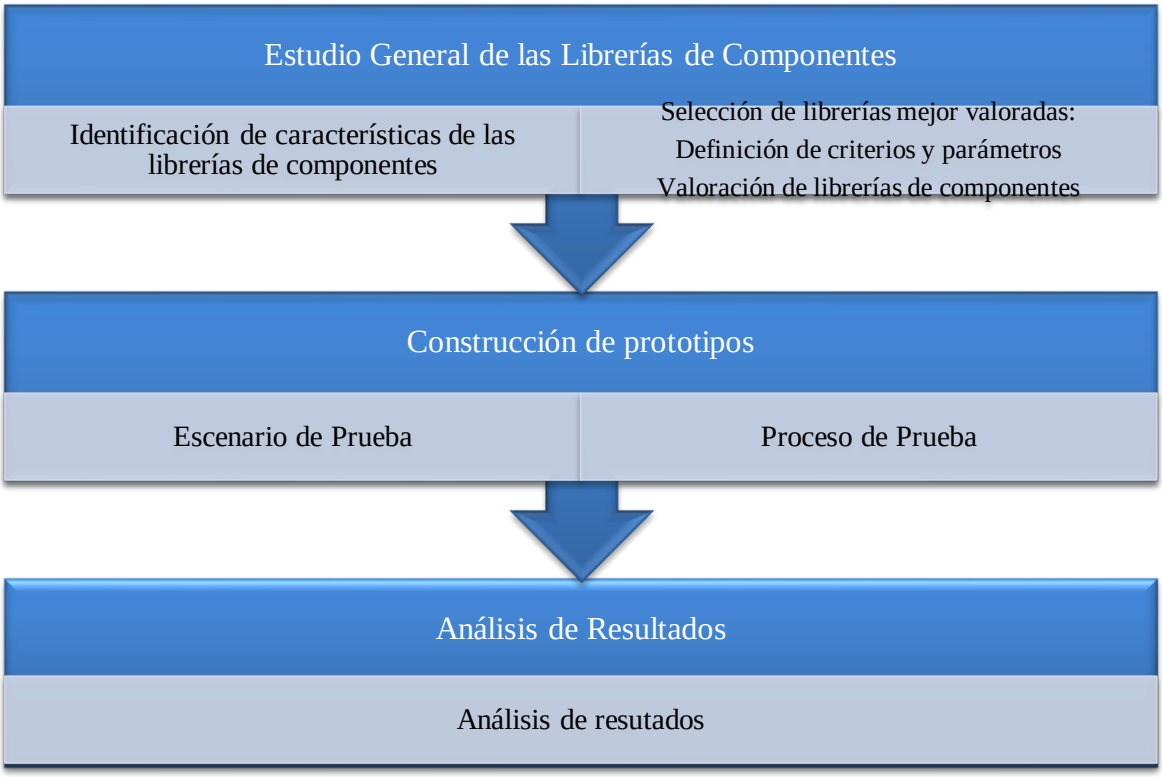
Fuente: Autoras

CAPÍTULO III

Análisis comparativo de librerías de componentes

A partir del estudio realizado en el capítulo anterior de cada una de las nueve librerías de componentes investigadas, se procederá a realizar un análisis comparativo de las mismas y así determinar cuál de ellas ayudará al desarrollo de las aplicaciones web enriquecidas de manera eficiente.

El proceso aplicado para el análisis comparativo es el siguiente:



3.1. Estudio general de las librerías de componentes

Como ya se ha visto anteriormente, JavaServer Faces posee una gran cantidad de librerías diferentes con componentes parecidos por lo que se hace a veces difícil escoger cual usar.

3.1.1. Identificación de características de las librerías de componentes

Se logró identificar varias características como se muestra en la tabla XI, definiéndolas para cada una de las librerías de componentes sometidas a estudio.

Tabla III. XI. Características generales librerías de componentes

	Richfaces	Icefaces	jQuery4jsf	Primefaces	Openfaces	Myfaces	ADF faces	RC faces	Prettyfaces
Tecnología	JSF	JSF	JSF	JSF	JSF	JSF	JSF	JSF	JSF
Diversidad de componentes	Alrededor de 70 componentes	70 componentes	Sólo componentes Ajax	117 componentes	Alrededor de 34 componentes	Alrededor de 20 componentes	Alrededor de 100 componentes	Alrededor de 56 componentes	Alrededor de 20 componentes
IE	Parcialmente	IE 6+	IE9 IE10	IE8 +	IE6	IE8+	IE7	IE6+	IE7 IE8
Firefox	V1.5 +	Todas las versiones	V16+	V14	V2	V2+	V2	V3+	V3
Safari	V3+	V 5.6	V3	V5 V6	V3	V3 V4 V5	V3 V4	V5	V3
Opera	V9.5+	V 9	V9	V12	V9.6	Parcialmente	V8+	V10	Si
Chrome	V1+	V25	Parcialmente	V22	Todas las versiones	V1 – V10	V0.2.149.30 +	V10	V4
JavaScript	Si	Si	Si	Si	Si	Si	Si	Si	Si
Soporte Ajax	Se debe hacer uso de Ajax4JSF, que no es tan transparente para el desarrollador, puesto que, además de introducir los componentes de RichFaces, se tiene que añadir componentes no visuales	Es transparente para el desarrollador, lo implementa de forma nativa en todos los componentes mediante la propiedad partialSubmi	Posee un conjunto de herramientas que hacen que sea transparente al desarrollador	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	Proporciona la capacidad de recargar componentes, invocar acciones servidor con Ajax	Posee conjunto de componentes Ajax habilitados cada uno con su propio conjunto de componentes de comportamiento especifíco	Utiliza un Tag para invocar eventos tipo Ajax	Limitaciones	Limitaciones

	de la librería Ajax4JSF.								
Componentes Ricos	No	Si	No	Si	No	No	No	No	No
Soporte Servidores	Apache Tomcat 5.5 y 6.0 JBoss 4.2.x y 5.1 Glassfish 2.1.1 Weblogic Server 10 IBM WebSphere Jetty	Glassfish Apache Tomcat JBoss	Apache Tomcat 6, Jboss 5, o GlassFish 3	Apache GlassFish Tomcat	Apache Tomcat 6.0 GlassFish v3.	Tomcat 5.5 Apache	Apache	Tomcat 5,5 a 6,0 WebLogic 10,3 JBoss 5.0 GlassGish V3	
Documentación	Si	Si	Parcialmente	Si	Parcialmente	Parcialmente	Parcialmente	Parcialmente	Casi nula

Fuente: Autoras

3.1.2. Selección de librerías mejor valoradas

Se realizará una comparación más específica entre las nueve librerías estudiadas, dependiendo de parámetros que se definirán a continuación.

A partir de ésta comparación se tomará en cuenta aquellas librerías de componentes con mayor valor para realizar los prototipos que ayudarán a definir la mejor para el desarrollo de aplicaciones web enriquecidas.

3.1.2.1. *Definición de criterios y parámetros de valoración*

En la tabla XII se especificará los criterios de valoración (valor, descripción y un rango de probabilidad) que se usará para cuantificar los parámetros que se establecerán para las librerías de componentes.

Tabla III. XII. Criterios de Valoración

Valor	Descripción	Rango de Probabilidad
1	Baja	**1% - 33%**
2	Media	**34% - 66%**
3	Alta	**67% - 100%**

3.1.2.2. Valoración de las librerías de componentes

Acorde con el estudio realizado en el capítulo anterior y con la comparación general se definirá una comparativa más detallada y explícita, tomando en cuenta las características más relevantes en cuanto al desarrollo para lo que se tomará en cuenta los parámetros e indicadores mostrados en la tabla XIII:

Tabla III. XIII. Parámetros e indicadores de comparación

Parámetro	Indicador
Librerías más usadas actualmente	Facilidad de Uso
	Diversidad de Componentes
	Compatibilidad con Navegadores
Librerías que han alcanzado mayor madurez a lo largo de su desarrollo	Facilidad para iniciar
	Documentación
	Rendimiento

Librerías más usadas actualmente

Se tomó en cuenta este parámetro debido a las tendencias tecnológicas que cada día avanzan dejando atrás aquellas librerías que se vuelven obsoletas al momento del desarrollo de aplicaciones. Para la identificación de las librerías más usadas actualmente se les dará un valor en base a los siguientes indicadores:

- *Facilidad de Uso:* Es necesario que las librerías de componentes permitan un manejo fácil y rápido para agilitar la programación de los desarrolladores.
- *Diversidad de Componentes:* Las librerías de componentes tienen una variedad de componentes que ayudan al desarrollador a elaborar páginas web de una manera más rápida y obtener variedad de componentes para realizar una misma tarea.
- *Compatibilidad con Navegadores:* Es necesario que las librerías de componentes sean capaces de poder ejecutarse sin distinción visual en cualquier navegador.

Librerías que han alcanzado mayor madurez a lo largo de su desarrollo

Se añadirá los siguientes indicadores que ayudarán a determinar las librerías que han alcanzado mayor madurez a lo largo de su desarrollo brindando al programador mejores herramientas al momento de realizar un proyecto:

- *Facilidad para iniciar.* Las librerías de componentes deben permitir al desarrollador iniciar cada uno de los componentes de una manera rápida, sin tener que incrementar librerías de inicio cada vez que se requiera usar dichas librerías.
- *Documentación:* Es necesario que cada librería de componentes posea la documentación necesaria, como tutoriales o ayudas on-line, para facilitar uso de las mismas.
- *Rendimiento:* Al momento de ejecutar un proyecto con ayuda de las librerías de componentes, éste debe tener respuestas, procesamiento de páginas, una tasa de transferencia de datos, etc., rápidas y eficaz.

En la Tabla XIV se mostrará la comparación y la respectiva valoración entre las librerías de componentes con cada uno de los indicadores mencionados anteriormente para obtener los valores de manera clasificada mediante las librerías más actuales.

Tabla III. XIV. Parámetros de evaluación, librerías más actuales

Librerías más Actuales					
Librería de Componentes	Facilidad de Uso	Diversidad de Componentes	Compatibilidad con Navegadores	Total	Promedio %
Richfaces	3	3	2	8	15,09
Icefaces	2	2	2	6	11,32
jQuery4jsf	1	1	1	3	5,66
Primefaces	3	3	3	9	16,98

Openfaces	2	2	2	6	11,32
Myfaces	2	2	2	6	11,32
ADF faces	2	2	1	5	9,43
RC faces	2	1	2	5	9,43
Prettyfaces	2	1	2	5	9,43

En base a la Tabla XIV, se observa que las tres librerías con mayores valores y porcentajes darán a conocer aquellas librerías más usadas actualmente, como se representa en los gráficos 17 y 18 en cuanto a los indicadores antes mencionados, y así se reconocerá aquellas librerías que brindan un manejo fácil y rápido, con una variedad de componentes para ayudar a la elaboración de páginas web, además de cuáles poseen la compatibilidad con navegadores y la capacidad de ejecutarse sin ninguna distinción visual.

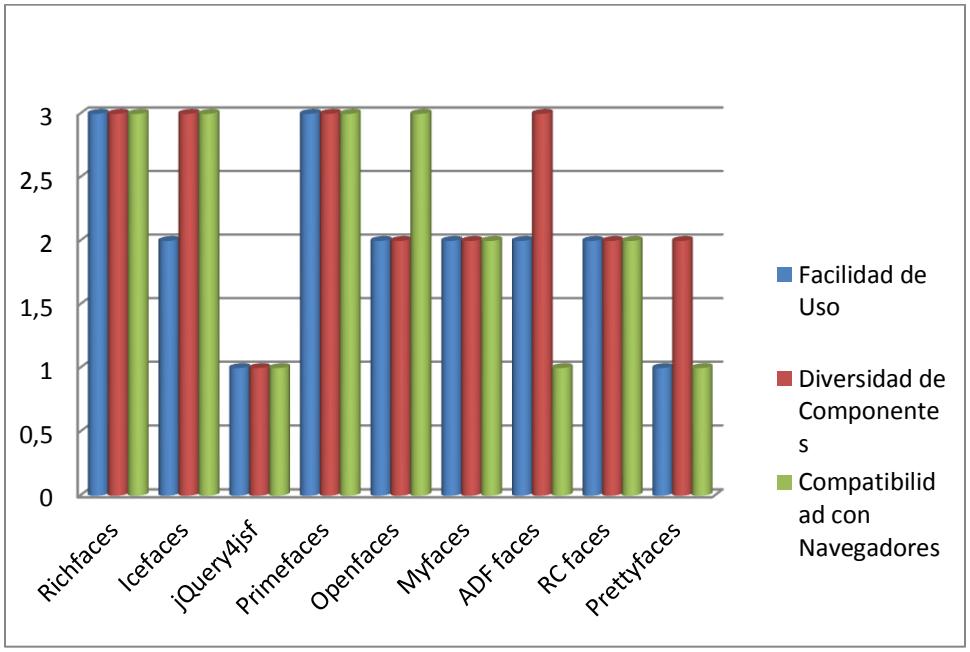


Figura III. 17. Librerías más actuales valorizadas según indicadores

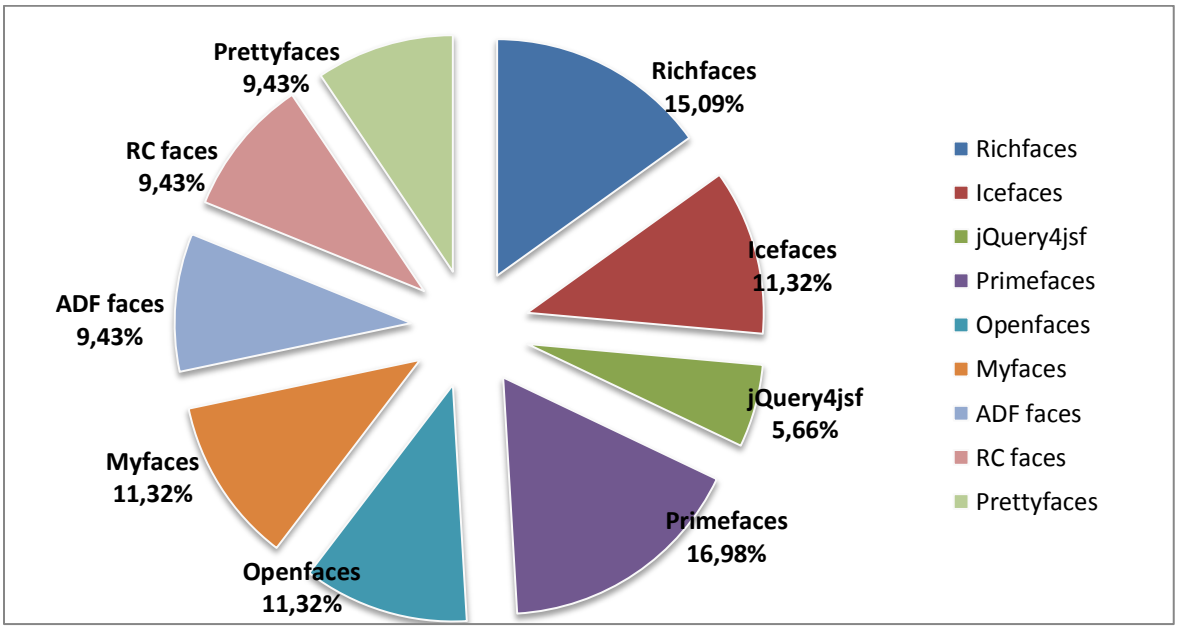


Figura III. 18. Librerías más actuales según valores en porcentajes

A partir de los valores obtenidos se tiene como resultado que las librerías Richfaces con el 15.09%, Icefaces con el 11.32% y Primefaces con el 16.98% son las más usadas actualmente.

En la Tabla XV se mostrará la comparación y la respectiva valoración entre las librerías de componentes con cada uno de los indicadores mencionados anteriormente para obtener los valores de manera clasificada mediante las librerías que han alcanzado mayor madurez.

Tabla III. XV. Parámetros de evaluación, librerías que han alcanzado mayor madurez

	Madurez en el desarrollo de Aplicaciones				
Librería de Componentes	Facilidad para iniciar	Documentación	Rendimiento	Total	Promedio %
Richfaces	3	3	2	8	14,81
Icefaces	2	3	2	7	12,96
jQuery4jsf	1	2	1	4	7,41
Primefaces	3	3	3	9	16,67
Openfaces	2	2	2	6	11,11
Myfaces	2	2	2	6	11,11
ADF faces	2	2	1	5	9,26

RC faces	2	2	1	5	9,26
Prettyfaces	1	1	2	4	7,41

De acuerdo con los resultados de la tabla XV se definirá aquellas tres librerías con mayor valor y porcentaje que darán a conocer las librerías de componentes que han alcanzado mayor madurez para el desarrollo de aplicaciones web enriquecidas, como se indica en los gráficos 19 y 20, partiendo de los indicadores de éste parámetro, se logró definir aquellas que permiten la facilidad para iniciar los componentes con un repositorio centralizado de librerías, así como la disponibilidad de documentación de cada librería, además de mejorar el rendimiento en cuanto a tiempos de respuesta procesamiento de páginas, etc.

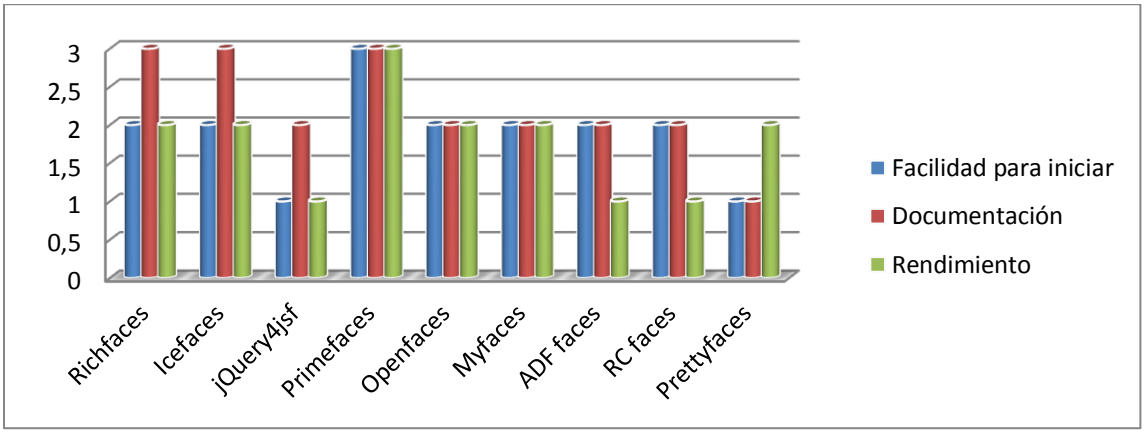


Figura III. 19. Librerías que han alcanzado mayor madurez valorizadas según indicadores

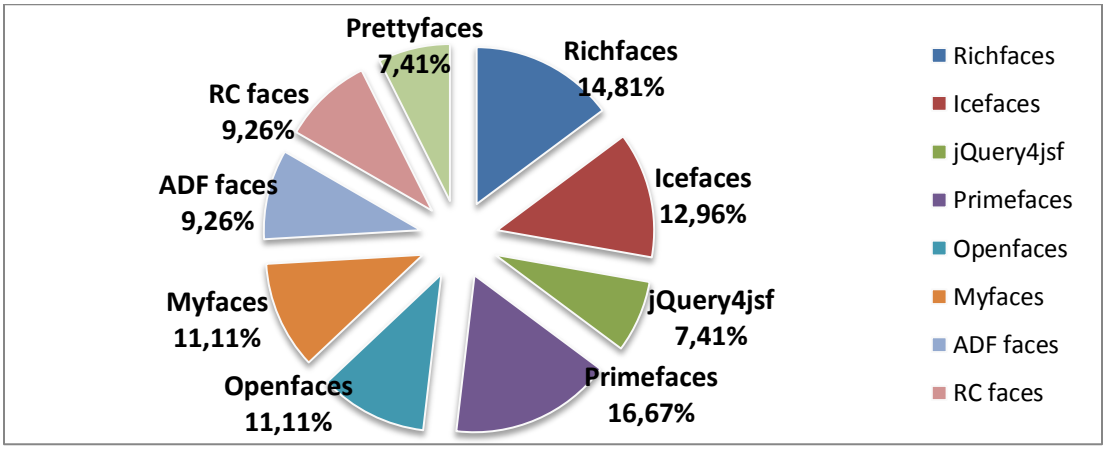


Figura III. 20. Librerías que han alcanzado mayor madurez según valores en porcentajes

Los gráficos 19 y 20 dan a conocer que las librerías Richfaces con el 14.81%, Icefaces con el 12.96% y Primefaces con el 16.67% han alcanzado mayor madurez para el desarrollo de aplicaciones web enriquecidas.

Se concluye que las tres librerías **Richfaces, Icefaces,** y **Primefaces** son las más usadas actualmente y además poseen la madurez para ser ampliamente adoptadas en el desarrollo de aplicaciones web enriquecidas.

3.2. Construcción de Prototipos

3.2.1. Escenario de prueba

Se analizarán las tres librerías, Richfaces, Icefaces y Primefaces para comprobar que reducen el tamaño de página y tamaño de respuestas AJAX, dando un mejor rendimiento y eficiencia. Usando la creación de DataTable y de la herramienta NeoLoad, como indica la figura 21:

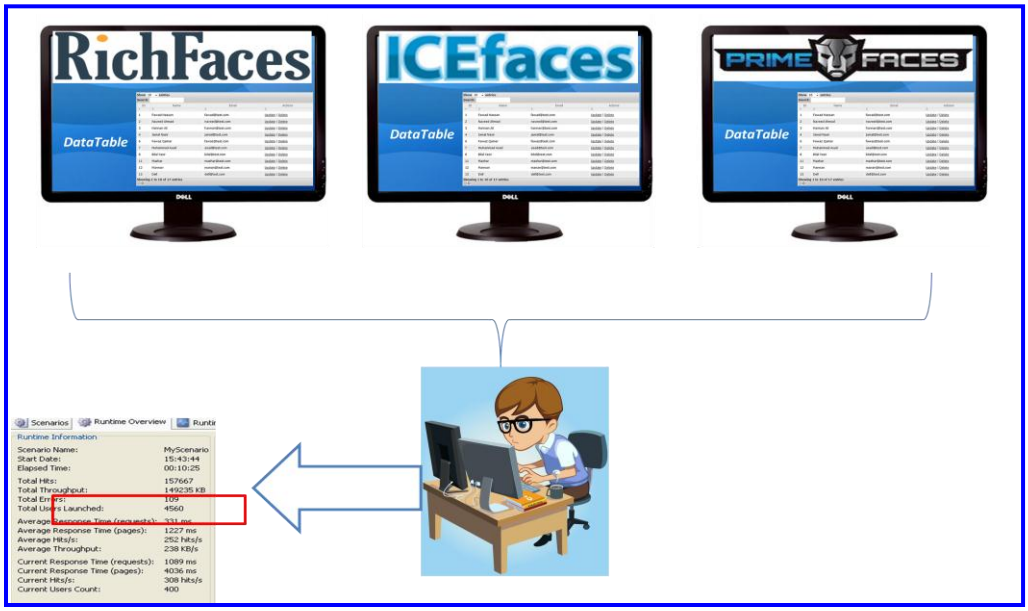


Figura III. 21: Escenario de pruebas

3.2.2. Proceso de prueba

Ya obtenidas las tres librerías se elaboró prototipos donde se utilizó el componente DataTable presente en las tres librerías. Se tomó la tabla *Color* que contiene información sobre los Colores de los Vehículos, dirigiéndose a la siguiente información: Código, Codificación y Nombre. El componente DataTable utiliza un AJAX para la paginación, que muestra 15 Colores por página.

Para realizar lo descrito anteriormente se utilizó líneas de código que se encuentran en el Anexo 1, que muestra los registros contenidos en la tabla especificando el diseño de la paginación.

Al momento de llevar a cabo las pruebas se utilizó la herramienta NeoLoad, la misma que permite obtener un sin número de parámetros como indica en la Figura 22, ayudará con la medición de la longitud de la página y respuestas AJAX.

Esta herramienta se aplicó a cada uno de los prototipos creados, que se detalla en el Anexo 2, de los cuales se obtuvo diferentes resultados los mismos que serán analizados más adelante.

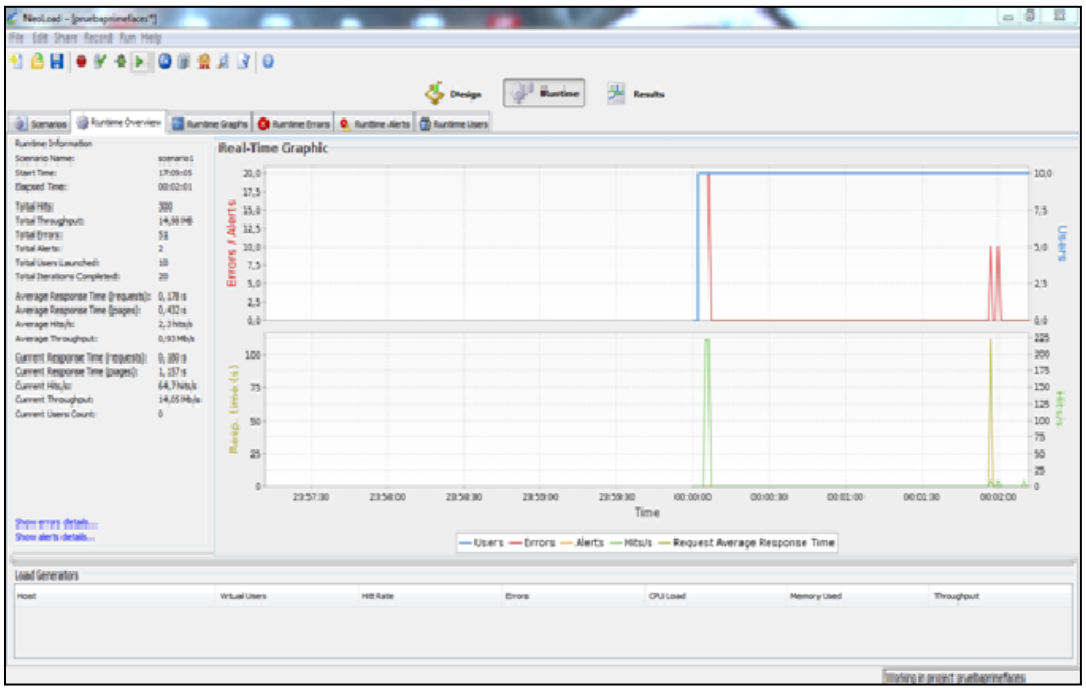


Figura III. 22. Resultados de pruebas en NeoLoad

3.3. Análisis de resultados

A partir del proceso de prueba especificado anteriormente se obtuvo los siguientes resultados:

Tamaño de Página

NoeLoad - richfaces HTML: 8KB JS: 275 KB CSS: 7KB Total: 290KB

NoeLoad - icefaces HTML: 22KB JS: 1275 KB CSS: 105KB Total: 1402KB

NoeLoad - primefaces HTML: 4KB JS:247 KB CSS:38KB Total:289KB

Tamaño de Respuesta Ajax

NoeLoad - richfaces HTML: 6KB JS: 0KB CSS: 0KB Total: 6KB

NoeLoad - icefaces HTML: 12KB JS: 0KB CSS: 0KB Total: 12KB

NoeLoad - primefaces HTML: 2KB JS: 0KB CSS: 0KB Total: 2KB

Los datos anteriores indican que en cuanto al tamaño de página ICEfaces es de mayor extensión. Icefaces y RichFaces no utilizan la "compresión" a JavaScript, es decir la eliminación de caracteres innecesarios como espacios en blanco, comentarios, y código delimitadores pero sin perder la funcionalidad de los mismos.

En lo que concierne a tamaños de respuesta AJAX Icefaces y RichFaces enviaron el paginador y la tabla juntas, la tabla que contiene los identificadores y clases con todos los campos. El bloque de código paginador de ICEfaces es superior a la codificación del DataTable.

En Primefaces se envió una única tabla. La paginación se actualiza saliendo del cliente. La tabla contiene sólo los identificadores y clases en una sola línea.

Después de haber obtenido los resultados tanto para lo que es el tamaño de página y la respuesta AJAX, en la tabla XVI basándose en la leyenda de valores[1] se realizará un cuadro comparativo entre las librerías Richfaces, Icefaces y Primefaces.

1 Leyenda de valores pag 78

Tabla III. XVI. Valores de Rendimiento y eficiencia (reducción tamaño de página y de respuesta Ajax) de las librerías Richfaces, Icefaces y Primefaces

Librería de Componentes	Reducción de página	Reducción de tamaño de respuesta AJAX	Total	Promedio %
RichFaces	2	2	4	30,77
IceFaces	2	1	3	23,08
Primefaces	3	3	6	46,15

Después de realizadas las respectivas valoraciones, en la figura 23 se mostrará que Primefaces alcanza la mayor valoración en cuanto a la reducción de tamaño de página y de respuestas AJAX.

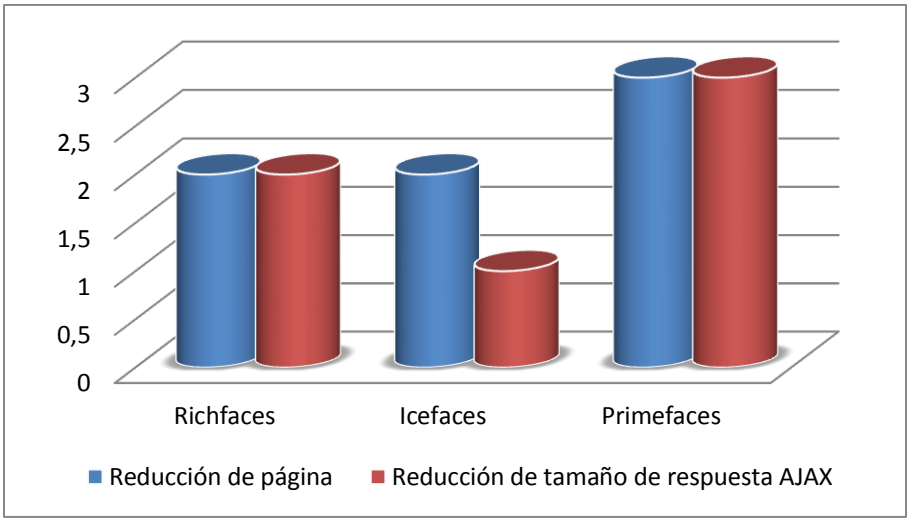


Figura III. 23. Valores de Rendimiento y Eficiencia en cuanto al tamaño de página y tamaño de respuesta Ajax

Así después de las comparaciones valorizadas se tiene como resultado el gráfico 23 donde porcentualmente Primefaces obtiene el mayor valor.

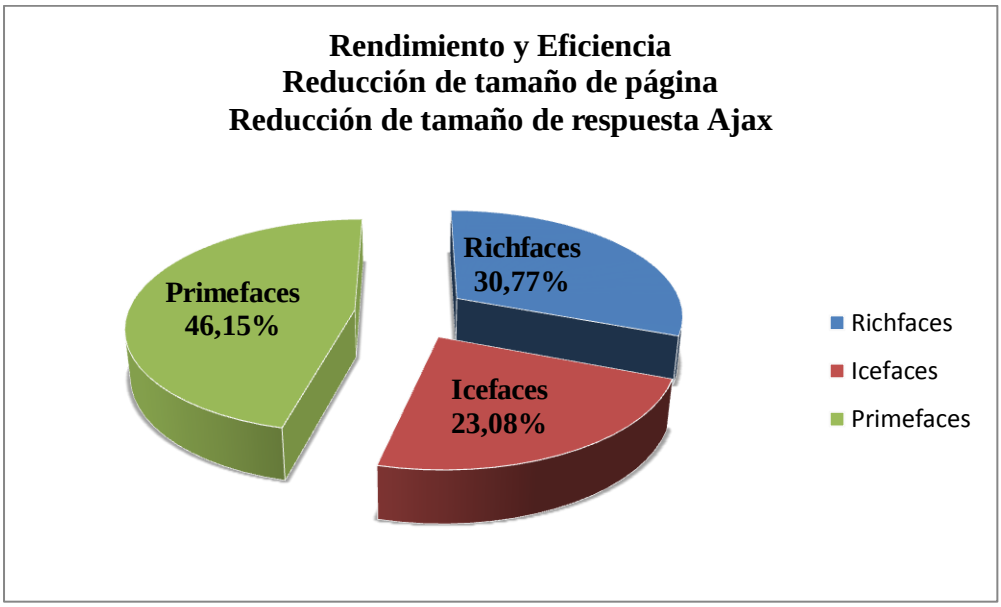


Figura III. 24. Valores de Rendimiento y Eficiencia en cuanto al tamaño de página y tamaño de respuesta Ajax en porcentajes por librería

Es así como al final se definirá que Primefaces tiene la mejor aplicación para DataTable. Icefaces demuestra claramente tener el peor desempeño con DataTable de todos los componentes probados. RichFaces estuvo entre los dos anteriores, sin embargo se aproxima a ser un framework de mejor desempeño.

Existen actualmente muchas librerías de componentes para JSF. El elegir utilizar una u otra dependerá en gran medida del número y la calidad de componentes que se ofrezca, la estabilidad en los cambios de versión, el disponer de una buena documentación, la utilización e integración con otros estándares, compatibilidad con otros navegadores etc., a partir de ello se puede decir que Primefaces cumple sobradamente con todas esas características, por lo que es una buena elección.

CAPÍTULO IV

Desarrollo del Sistema Control Jefatura de Movilización (SCJM)

Como se puede apreciar en el capitulo anterior Primefaces es la librería de componentes más adecuada para el desarrollo de aplicaciones web enriquecidas.

Por dicha razón ésta librería de componentes se usará para desarrollar el Sistema Control Jefatura de Movilización (SCJM) con una metodología de desarrollo de software eXtreme Programming(XP).

El sistema se desarrolló en el Departamento de DESITEL y lo utilizará el Departamento de Movilización de la ESPOCH, el mismo que ayudará a llevar un control automatizado de todos los procesos que se llevan a cabo en dicho departamento.

Todos estos temas serán tratados en el transcurso del presente capítulo, definiendo conceptos, características de la metodología y el desarrollo del sistema en si.

4.1.Metodología XP

La Programación Extrema o XP nace oficialmente hace cinco años fundada por Kent Beck, es el más destacado de los procesos ágiles de desarrollo de software. La programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad.

Los defensores de la XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

Esta metodología de desarrollo de software posee cuatro características básicas que debe reunir el programador XP que son: la simplicidad, la comunicación y la retroalimentación o reutilización del código desarrollado (reciclado de código).

El ciclo de vida de XP se define en cuatro tareas fundamentales como se indica en la Figura 25.

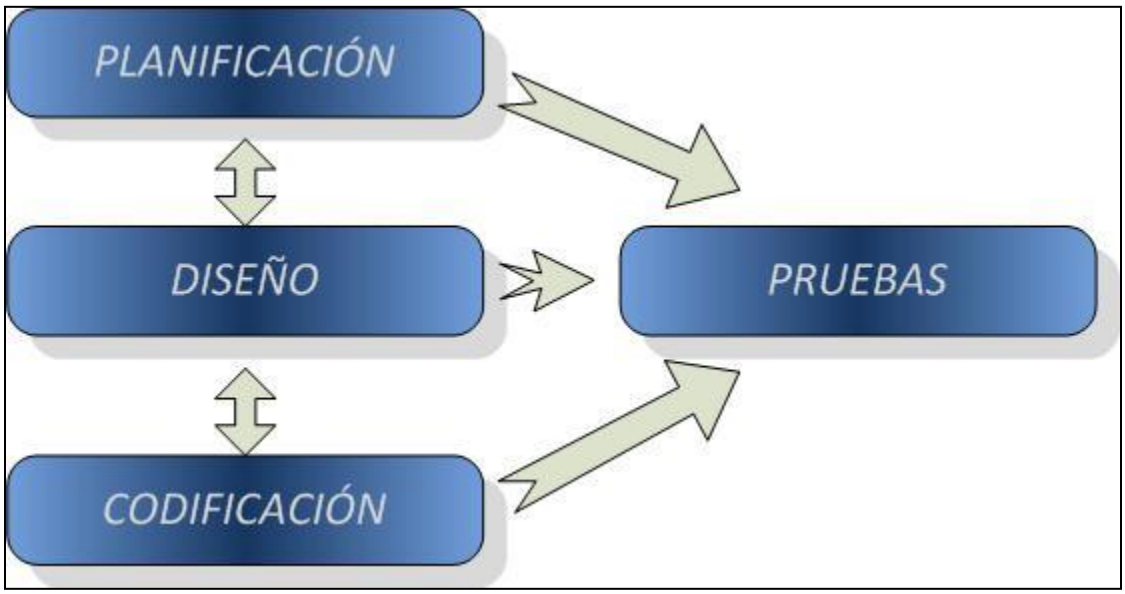


Figura IV. 25. Ciclo de vida XP

4.2.Desarrollo del Sistema

4.2.1. Gestión del proyecto

4.2.1.1.Planificación del proyecto

Esta planificación la realizamos al inicio de éste capítulo, tras estudiar el problema y reunir los requerimientos necesarios. De esta redacción inicial de historias de usuario se realizó una planificación inicial y posteriormente fue cambiada a lo largo del mismo, eliminando o cambiando historias de usuario, a medida que se tenía una concepción más clara del problema.

4.2.1.2.Integrantes y roles

Teniendo en cuenta la participación tanto de los Jefes de Proyecto, como de los usuarios y desarrolladores, se formará así el equipo encargado de la implementación de software. Esto

implicará que los diseños deberán ser claros y sencillos, los usuarios deberán disponer de versiones operativas cuanto antes para poder participar en el proceso creativo mediante sus sugerencias y aportaciones, el potenciar al máximo el trabajo en equipo es fundamental para el desarrollo del sistema, dicho equipo de trabajo se ve ilustrado en la Tabla XVII definiendo Integrantes y Roles:

Tabla IV. XVII. Integrantes y Roles

Miembro	Grupo	Roles XP	Metodología
Karla Sosa	Tesistas	Rastreador, Testeador, Programador	**Xp**
Eulalia Carrillo	Tesistas	Rastreador, Programador, Testeador.	
Ing. Ivonne Rodriguez **Ing. Diego Palacios** **Ing. Anibal Herrera**	**Consultor**	**Entrenador**	

4.2.1.3.Prototipos

Las interfaces de usuario son las interfaces más importantes ya que de esta dependerá el entendimiento fácil y rápido por parte del usuario al comenzar a manipular el sistema, la amigabilidad de la interfaz reside en el uso de ventanas, cuadros de diálogo, gráficos, etc. Se pretende lograr una interfaz sencilla y funcional, con un alto grado de compresión, por estas razones se crearon prototipos generales del sistema

A continuación una breve descripción del proceso principal del proyecto:

- Indicándose en la Figura 26 el inicio de sesión de usuarios:

Figura IV. 26. Acceso de Usuarios

- En la figura 27 se muestra la pantalla de creación de una solicitud:

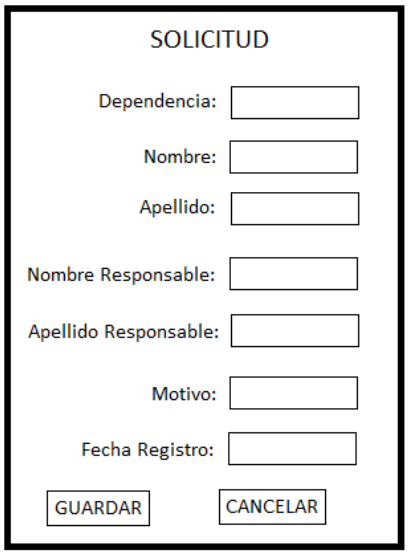


Figura IV. 27. Gestión Solicitud

- Una pantalla para describir datos donde a una solicitud ya creada se le asigne un conductor, una dependencia y un vehículo, como se muestra en la figura 28:

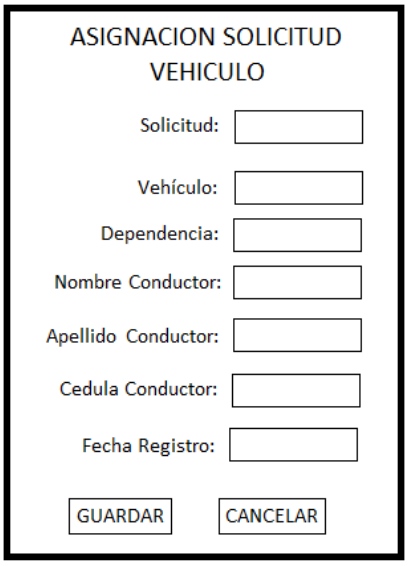


Figura IV. 28. Gestión Asignación

- Al momento de crear una nueva orden de tanqueo, se muestra una pantalla en la que se pide los datos necesarios para dicha actividad como se muestra a continuación en la figura 29:

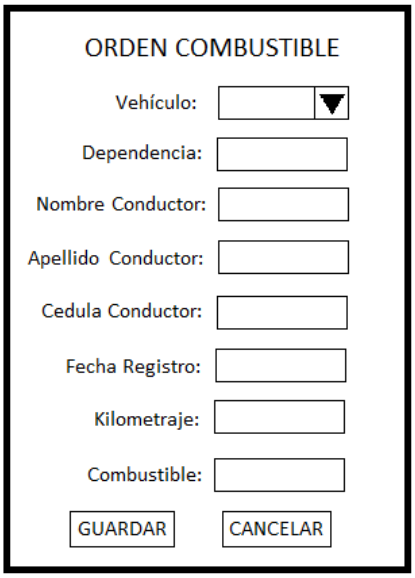


Figura IV. 29. Gestión Orden Combustible

- Después de realizado el despacho de combustible en la gasolinera, se ingresará al sistema los datos del mismo, como sse indica la figura 30:

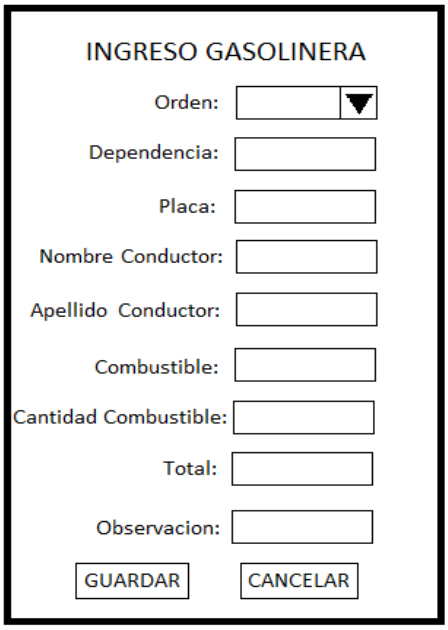


Figura IV. 30. Gestión Gasolinera

4.2.1.4.Historias de usuarios

Las historias de usuario tienen como propósito ver las necesidades del sistema; por tanto serán descripciones cortas y escritas en el lenguaje del usuario, sin terminología técnica, además proporcionarán los detalles sobre la estimación del riesgo y cuánto tiempo conllevará la implementación de dicha historia de usuario.

Una historia se divide en varias tareas planificables y medibles en su estado de realización, estas forman parte de una iteración, este plan indicará, por tanto, diferentes iteraciones del sistema y que se debe implementar en cada una de ellas. La realización de este plan debe tener en cuenta en la medida de lo posible, las prioridades de los usuarios, para satisfacerles en mayor medida, como se indica en la Tabla XVIII:

Tabla IV. XVIII. Historias de Usuarios

N°	NOMBRE	PRIORIDAD	RIESGO	ESFUERZO	ITERACION
1	Control de Acceso de Usuarios	Alta	Alto	Medio	1
2	Control de conductores	Alta	Alto	Alto	1
3	Control de vehículos	Alta	Alto	Alto	1
4	Control de recorridos	Alta	Alto	Medio	1
5	Control de la seguridad	Alta	Alto	Alto	2
6	Control Conductor-Dependencia-Vehículo	Alta	Alto	Alto	2
7	Control solicitud	Alta	Alto	Alto	2
8	Control Asignación	Alta	Alto	Alto	2
9	Control orden combustible	Alta	Alto	Alto	2
10	Control ingreso gasolinera	Alta	Alto	Alto	2
11	Termino de Proceso	Alta	Alto	Alto	2
12	Emisión de reportes	Alta	Alto	Alto	3
13	Emisión de tickets para la orden y el ingreso de gasolinera	Alta	Alto	Alto	3

4.2.1.5.Plan de entregas

El plan de entregas se usará para crear los planes de iteración para cada iteración. Es en este momento cuando los técnicos tomarán las decisiones. En esta reunión estarán presentes tanto desarrolladores como usuarios.

Con cada historia de usuario previamente evaluada en tiempo de desarrollo ideal, el usuario las agrupará en orden de importancia.

De esta forma se puede trazar el plan de entregas en función de estos dos parámetros: *tiempo de desarrollo ideal y grado de importancia para el usuario*. Las iteraciones individuales son planificadas en detalle justo antes de que comience cada iteración como se puede apreciar en las siguientes tablas y figuras.

Iteración 1

Tabla IV. XIX. Plan de Entrega Iteración 1.

Historia de Usuario	Duración en semanas
Control de Acceso de Usuarios	2
Control de conductores	3
Control de vehículos	3
Control de recorridos	3

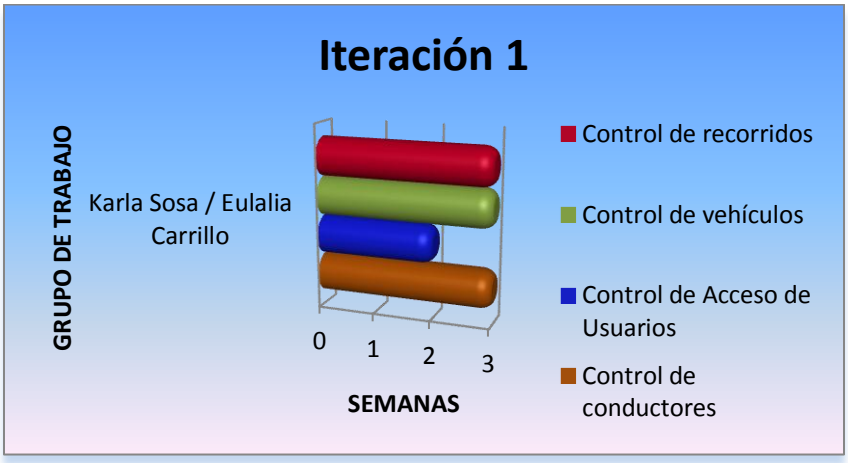


Figura IV. 31. Plan de Entrega. Iteración 1.

Iteración 2

Tabla IV. XX. Plan de Entrega Iteración 2.

Historia de Usuario	Duración en semanas
Control de la seguridad	2
Control Conductor-Dependencia-Vehículo	2
Control solicitud	2
Control Asignación	3
Control orden combustible	3
Control ingreso gasolinera	3
Termino de proceso	2

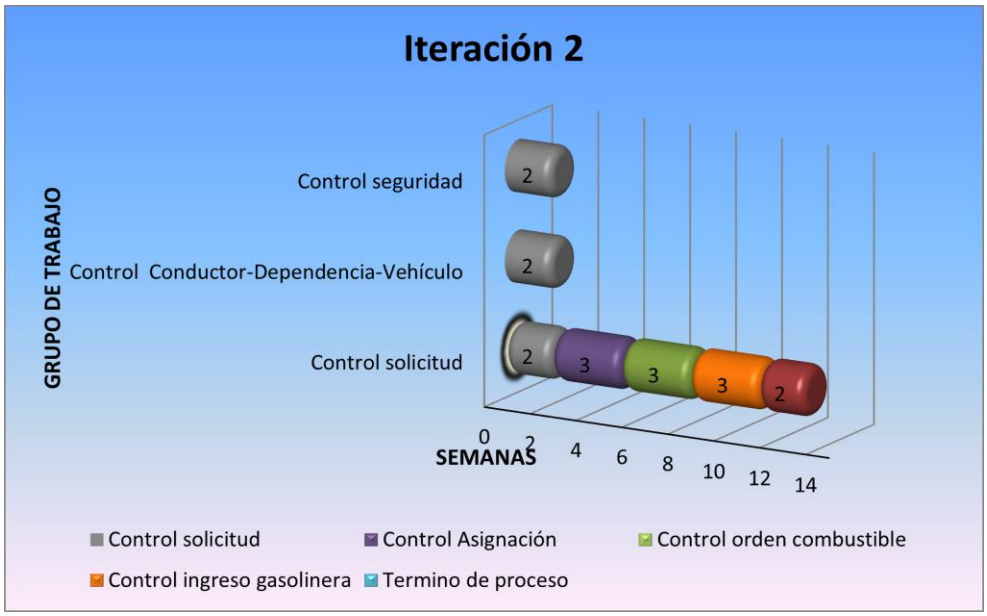


Figura IV. 32. Plan de Entrega. Iteración 2.

Iteración 3

Tabla IV. XXI. Plan de Entrega Iteración 3.

Historia de Usuario	Duración en semanas
Emisión de reportes	**2**
Emisión de tickets para la orden y el ingreso de gasolinera	**2**

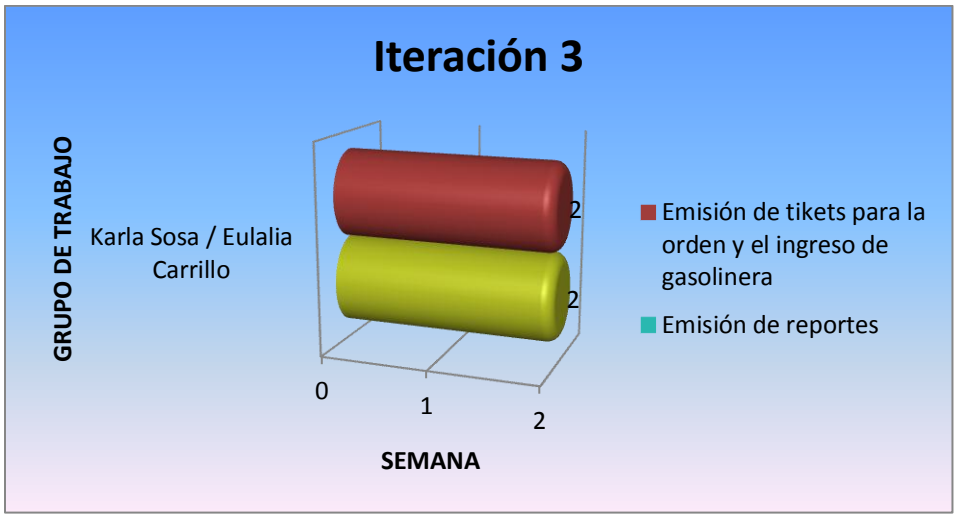


Figura IV. 33. Plan de Entrega. Iteración 3.

4.2.1.6.Incidencia

Iteración primera: Se tratara de tener preparadas las funcionalidades básicas relacionadas con el usuario. Tras esta fase nos ha quedado claro que es prácticamente imposible crear una planificación inmutable, se debe esperar que la planificación mostrada en principio varíe así como la desaparición y sustitución de algunas historias de usuario. El ensamblaje de un grupo de trabajo es una labor larga y laboriosa, pequeños problemas como la selección de herramientas y unificación de horarios se convierten en principal piedra de toque de esta fase.

Iteración segunda: La importancia del usuario final ha quedado demostrada como máxima, ya que la visión de los miembros del equipo puede llegar a tener una interpretación distinta a la hoja de usuario.

Iteración tercera: Siendo esta la ultima iteración se pretende entregar el producto acabado con todas las funcionalidades propuestas por el usuario.

A continuación se describirá de una manera más detallada, iteración por iteración y cada una de las historias de usuario que las conforman (Ver tablas XXII - XXXIV).

Iteración 1

Tabla IV. XXII. Iteración 1. Historia 1.

Historia de Usuario	
Número: 1	***Usuario:*** Administrador, Vicerrector, Jefe de Dependencia, Jefe de Movilización, Conductor, Delegado de Gasolinera
Nombre historia: Control de Acceso de Usuarios.	
Prioridad en negocio: Alta	*Riesgo en desarrollo**: Alto***
Esfuerzo: **Medio**	*Iteración asignada: 1*
***Programador responsable**: Karla Sosa/ Eulalia Carrillo*	
***Descripción**: Antes de iniciar la aplicación se solicita la cuenta de usuario y su clave para que tenga acceso a los datos que corresponden a su tipo de usuario.*	
***Observaciones**: Hay seis tipos de usuarios: Administrador, Vicerrector, Jefe de Dependencia, Jefe de Movilización, Conductor, Delegado de Gasolinera con distintos permisos de acceso a los menús dependiendo de las funciones que les corresponden.*	

Tabla IV. XXIII. Iteración 1. Historia 2.

Historia de Usuario	
Número: 2	***Usuario**: Jefe de Movilización*
Nombre historia: **Control de Conductores**	
Prioridad en negocio: Alta	***Riesgo en desarrollo**: Alto*
Esfuerzo: Alto	***Iteración asignada**: 1*
***Programador responsable:** Karla Sosa/ Eulalia Carrillo*	
***Descripción**: Los usuarios deben autenticarse en el sistema. En el cual deberá escoger el rol ADMINISTRADOR MOVILIZACION, en el mismo que se encuentra la opción Gestión Conductor en el cual nos presenta las opciones: Nuevo conductor y Gestión Licencias*	
***Observaciones**: Para insertar un nuevo conductor se debe escoger el tipo de licencia con el cual el Conductor cuenta*	

Tabla IV. XXIV. Iteración 1. Historia 3

Historia de Usuario	
Número: 3	***Usuario:*** Jefe de Movilización
Nombre historia: **Control Vehículos**	
Prioridad en negocio: Alta	***Riesgo en desarrollo:*** *Alto*
Esfuerzo: Alto	***Iteración asignada****: 1*
Programador responsable*: Karla Sosa/ Eulalia Carrillo*	
Descripción*: El usuario debe autenticarse en el sistema. En el cual deberá escoger el rol ADMINISTRADOR MOVILIZACION VEHICULOS, en el se encuentras las siguientes opciones: Características, Nuevo, Actualización Datos, Reporte del Vehículo respectivamente mientras que Consultas Caducidad y Alertas de Caducidad tienen que ver con el Soat, Seguro y Matrícula del Vehículo.*	
Observaciones*: Para insertar un nuevo Vehículo se debe tomar en cuenta todas las características y datos correspondientes a un Vehículo.*	

Tabla IV. XXV. Iteración 1. Historia 4

Historia de Usuario	
Número: 4	**Usuario**: Jefe de Movilización, Conductor
Nombre historia: Control Recorridos	
Prioridad en negocio: Alta	**Riesgo en desarrollo: Alto**
Esfuerzo: **Alto**	**Iteración asignada:** 1
Programador responsable: Karla Sosa/ Eulalia Carrillo	
Descripción: *Jefe de Movilización debe autenticarse en el sistema. En el cual deberá escoger el rol ADMINISTRADOR MOVILIZACION, en el mismo q se encuentra la opción Gestión Rutas Diarias en el se podrá controlar el ingreso de las rutas diarias de todos los Conductores.* *Conductor deben autenticarse en el sistema con su cuenta. En el cual ya tiene definido su rol y podrá escoger la opción Ver mis Datos y Nueva Ruta Diaria*	
Observaciones:	

Iteración 2

Tabla IV. XXVI. Iteración 2. Historia 5

Historia de Usuario	
Número: 5	***Usuario**: Administrador*
***Nombre historia:** Control de la seguridad*	
Prioridad en negocio: Alta	***Riesgo en desarrollo**: Alto*
Esfuerzo: Alto	***Iteración asignada**: 2*
***Programador responsable:** Karla Sosa/ Eulalia Carrillo*	
***Descripción**: El administrador una vez registrado en el sistema, éste tendrá acceso a todo el sistema en general y en especial deberá administrar todo el rol ADMINISTRADOR MASTER*	
Observaciones:	

Tabla IV. XXVII. Iteración 2. Historia 6

Historia de Usuario	
Número: 6	*Usuario:* ***Jefe Movilización***
Nombre historia: **Control Conductor-Dependencia-Vehículo**	
Prioridad en negocio: **Alta**	*Riesgo en desarrollo:* ***Alto***
Esfuerzo: **Alto**	*Iteración asignada:* ***2***
Programador responsable: ***Karla Sosa/ Eulalia Carrillo***	
***Descripción**: El usuario una vez registrado en el sistema, en el cual deberá escoger el rol ADMINISTRADOR MOVILIZACION VEHICULOS, se deberá escoger el Submenú Gestión Conductor-Dependencia-Vehículo*	
***Observaciones**: Para asignar un conductor a una dependencia con su conductor estos ya deberán estar ingresados en l base de dato.*	

Tabla IV. XXVIII. Iteración 2. Historia 7

Historia de Usuario	
Número:7	***Usuario****: Jefe Dependencia*
Nombre historia: *Control Solicitud*	
Prioridad en negocio: Alta	***Riesgo en desarrollo****: Alto*
Esfuerzo: Alto	***Iteración asignada****: 2*
Programador responsable*: Karla Sosa/ Eulalia Carrillo*	
Descripción*: El Jefe Dependencia una vez registrado en el sistema, ésta ya tiene definido su rol y podrá escoger la opción Creación Solicitud Movilización y procederá a crear una nueva Solicitud*	
Observaciones*: Jefe de Dependencia dará el visto bueno y enviará la solicitud, y el Vicerrector deberá aprobar las solicitudes que se creen.*	

Tabla IV. XXIX. Iteración 2. Historia 8

Historia de Usuario	
Número: 8	***Usuario****: Vicerrector, Jefe de Movilización*
Nombre historia: *Control Asignación*	
Prioridad en negocio: Alta	***Riesgo en desarrollo:*** *Alto*
Esfuerzo: Alto	***Iteración asignada:*** *2*
Programador responsable: *Karla Sosa/ Eulalia Carrillo*	
Descripción*: Una vez registrado en el sistema, éste tendrá acceso al rol ASIGNACION SOLICITUD-VEHICULO en la cual deberá asignar a una solicitud determinada el vehículo con la cual se realizara dicha solicitud*	
Observaciones*: Esta historia la realizara el Vicerrector, en caso de ausencia la tarea la realizara el Jefe de Movilización y se podrá crear una asignación siempre q este aprobada la solicitud*	

Tabla IV. XXX. Iteración 2. Historia 9

Historia de Usuario	
Número: 9	***Usuario****: Jefe de Movilización*
Nombre historia*: Control Orden Combustible*	
Prioridad en negocio: Alta	***Riesgo en desarrollo****: Alto*
Esfuerzo: Alto	***Iteración asignada****: 2*
Programador responsable*: Karla Sosa/ Eulalia Carrillo*	
Descripción*: Jefe de Movilización debe autenticarse en el sistema. En el cual deberá escoger el rol ADMINISTRADOR MOVILIZACION, en el mismo q se encuentra la opción Gestión Tanqueo en la cual se creara una Orden para Despacho de Combustible*	
Observaciones*: Se podrá crear una orden de combustible cuando ya esté asignada una solicitud un vehículo.*	

Tabla IV. XXXI. Iteración 2. Historia 10

Historia de Usuario	
Número: 10	*Usuario:* ***Delegado de la Gasolinera***
Nombre historia: *Control Ingreso Gasolinera*	
Prioridad en negocio: Alta	***Riesgo en desarrollo:*** *Alto*
Esfuerzo: Alto	***Iteración asignada: 2***
Programador responsable*: Karla Sosa/ Eulalia Carrillo*	
Descripción*: El Delegado de la Gasolinera una vez registrado en el sistema, verificara que se emitió la orden de combustible y procederá al despacho de la misma, y así creando una orden de despacho de combustible.*	
Observaciones*: Para realizar esta actividad, deberán existir ordenes de combustibles*	

Tabla IV. XXXII. Iteración 2. Historia 11

Historia de Usuario	
Número: 11	***Usuario****: Jefe de Movilización*
Nombre historia: *Termino de proceso*	
Prioridad en negocio: Alta	***Riesgo en desarrollo****: Alto*
Esfuerzo: Alto	***Iteración asignada****: 2*
Programador responsable*: Karla Sosa/ Eulalia Carrillo*	
Descripción*: El usuario debe autenticarse en el sistema. En el cual deberá escoger el rol ADMINISTRADOR MOVILIZACION, en el mismo q se encuentra la opción Gestión Tanqueo en la cual ingresaremos al vinculo Entrega Vehículo*	
Observaciones*: Para realizar esta actividad, el conductor debe reportar su arribo a los patios de parqueo en la ESPOCH del recorrido realizado*	

Iteración 3

Tabla IV. XXXIII. Iteración 3. Historia 12

Historia de Usuario	
Número: 12	***Usuario:*** *Jefe de Movilización*
Nombre historia: *Emisión de reportes*	
Prioridad en negocio: Alta	***Riesgo en desarrollo****: Alto*
Esfuerzo: Alto	***Iteración asignada****: 3*
Programador responsable: *Karla Sosa/ Eulalia Carrillo*	
Descripción*: Jefe de Movilización debe autenticarse en el sistema. En el cual deberá escoger el rol ADMINISTRADOR MOVILIZACION, en el mismo q se encuentra la opción Reportes en la cual encontrara un listado de los diferentes reportes que necesite*	
Observaciones:	

Tabla IV. XXXIV. Iteración 3. Historia 13

	Historia de Usuario
Número: 13	***Usuario****: Jefe de Movilización, Delegado de la Gasolinera*
Nombre historia: Emisión de tickets para la orden y el ingreso de gasolinera	
Prioridad en negocio: **Alta**	***Riesgo en desarrollo:*** *Alto*
Esfuerzo: Alto	***Iteración asignada:*** *3*
Programador responsable: *Karla Sosa/ Eulalia Carrillo*	
Descripción*: Los usuarios deben autenticarse en el sistema con sus cuentas respectivamente y al generar las Ordenes de Combustible se deberá imprimir un tickets de constancia así como en la gasolinera de un tickets de despacho de combustible*	
Observaciones:	

4.2.1.7.Actividades

Las actividades de nuestro sistema fueron divididas en varios procesos reflejados en los siguientes diagramas o flujos de procesos:

- Proceso Vehículo, proceso en el cual se desarrolla todo lo que conlleva a los datos de un vehículo como se muestra en la figura 34, ingreso de datos, reportes, etc.

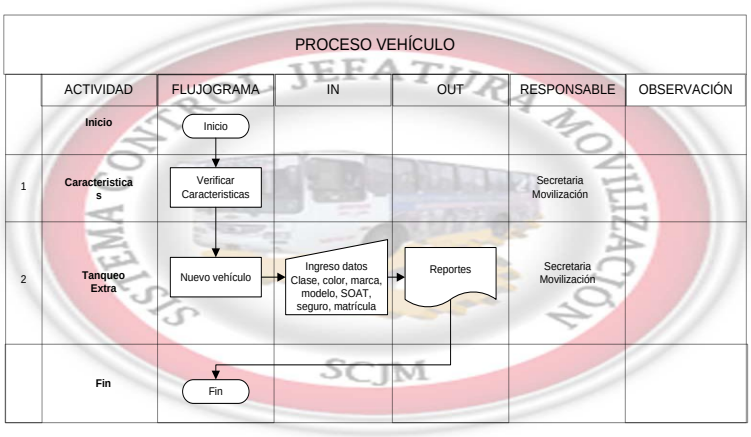


Figura IV. 34. Diagrama de proceso del Vehículo

- Proceso Conductor, proceso en el cual se desarrolla todo lo que conlleva a los datos de un conductor como se indica en la figura 35, ingreso de datos, reportes, etc.

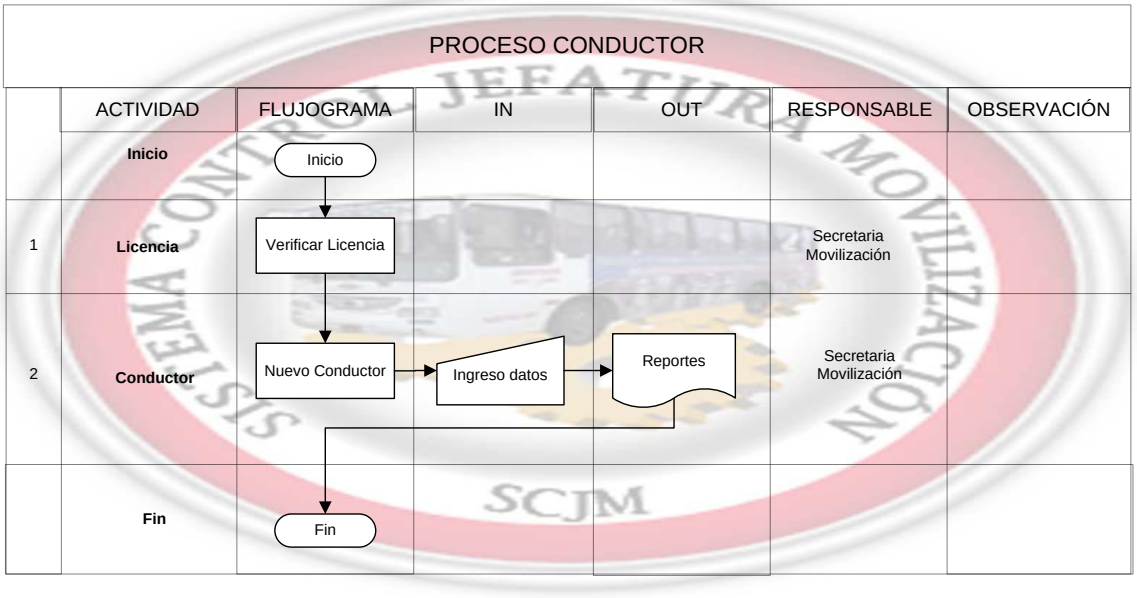


Figura IV. 35. Diagrama de proceso del Conductor

- Proceso Dependencia, proceso en el cual se desarrolla todo lo que conlleva a los datos de una dependencia como se muestra en la figura 36, ingreso de datos, reportes, etc.

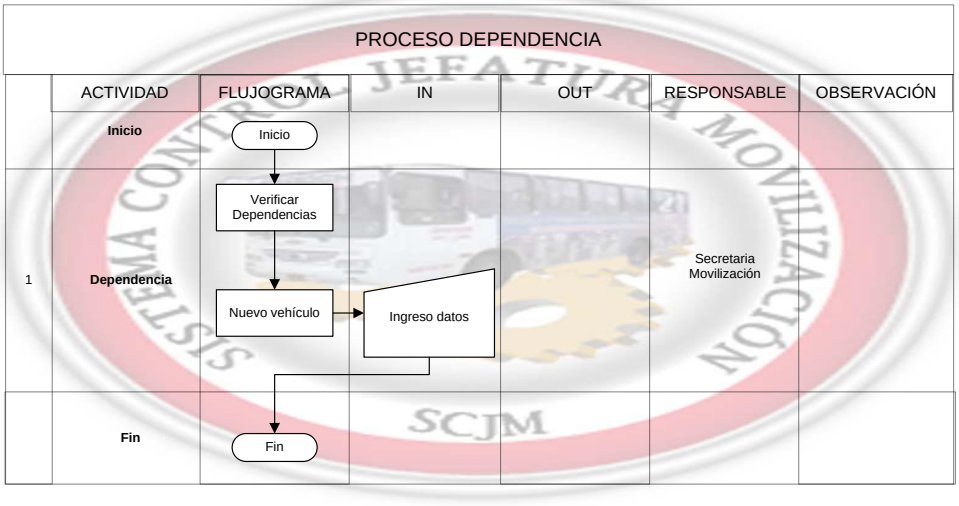


Figura IV. 36. Diagrama de proceso de la Dependencia

- Proceso Asignar Dependencia Vehículo, donde se detalla todo lo que concierne a una nueva asignación como se muestra en la figura 37, ingreso de datos, reportes, etc.

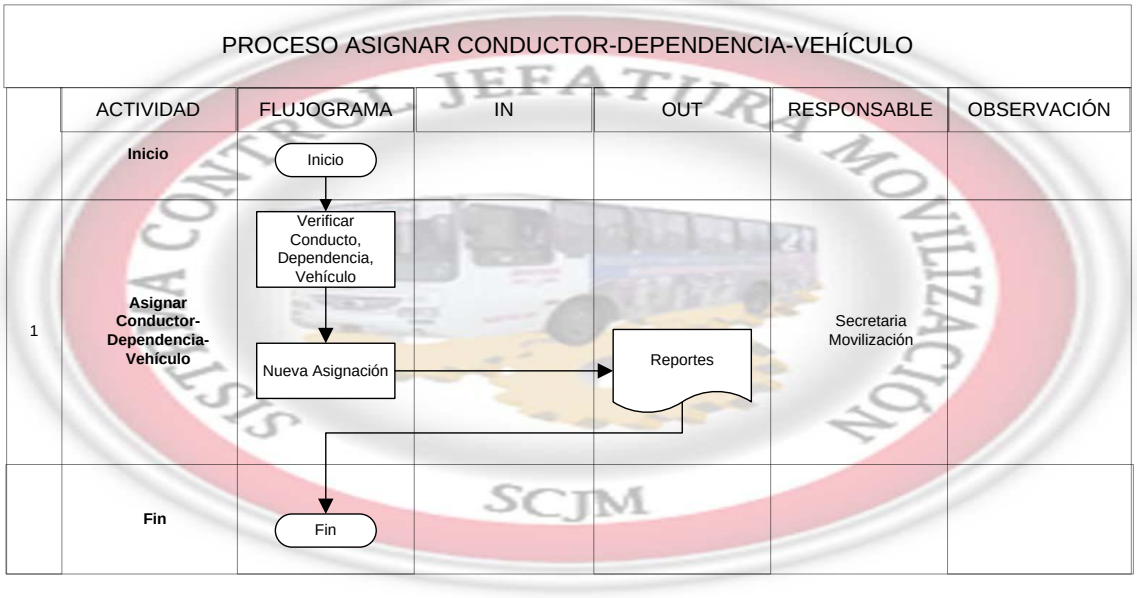


Figura IV. 37. Diagrama de proceso del Conductor-Dependencia-Vehículo

- Proceso Recorrido, donde se muestra la creación de un nuevo recorrido, de un tanqueo extra como se muestra en la figura 38, ingreso de datos, reportes, etc.

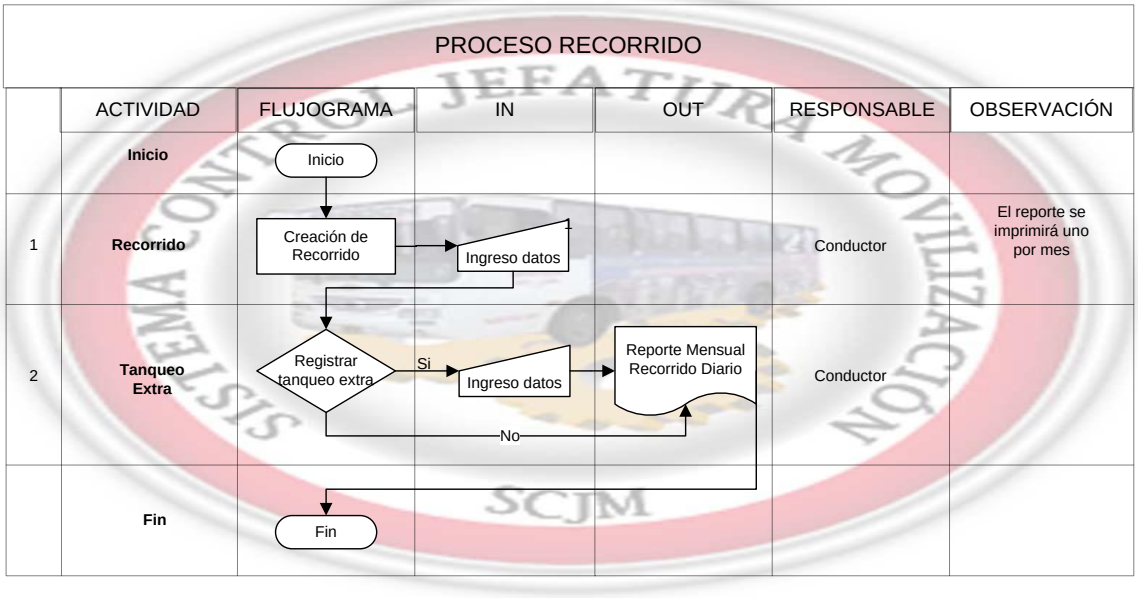


Figura IV. 38. Diagrama de proceso de Recorrido del Conductor

- Proceso Tanqueo Gasolina, proceso principal del sistema en el cual se detalla solicitudes, asignaciones, órdenes tal y como indica figura 39, ingreso de datos, emisión de tickets, etc.

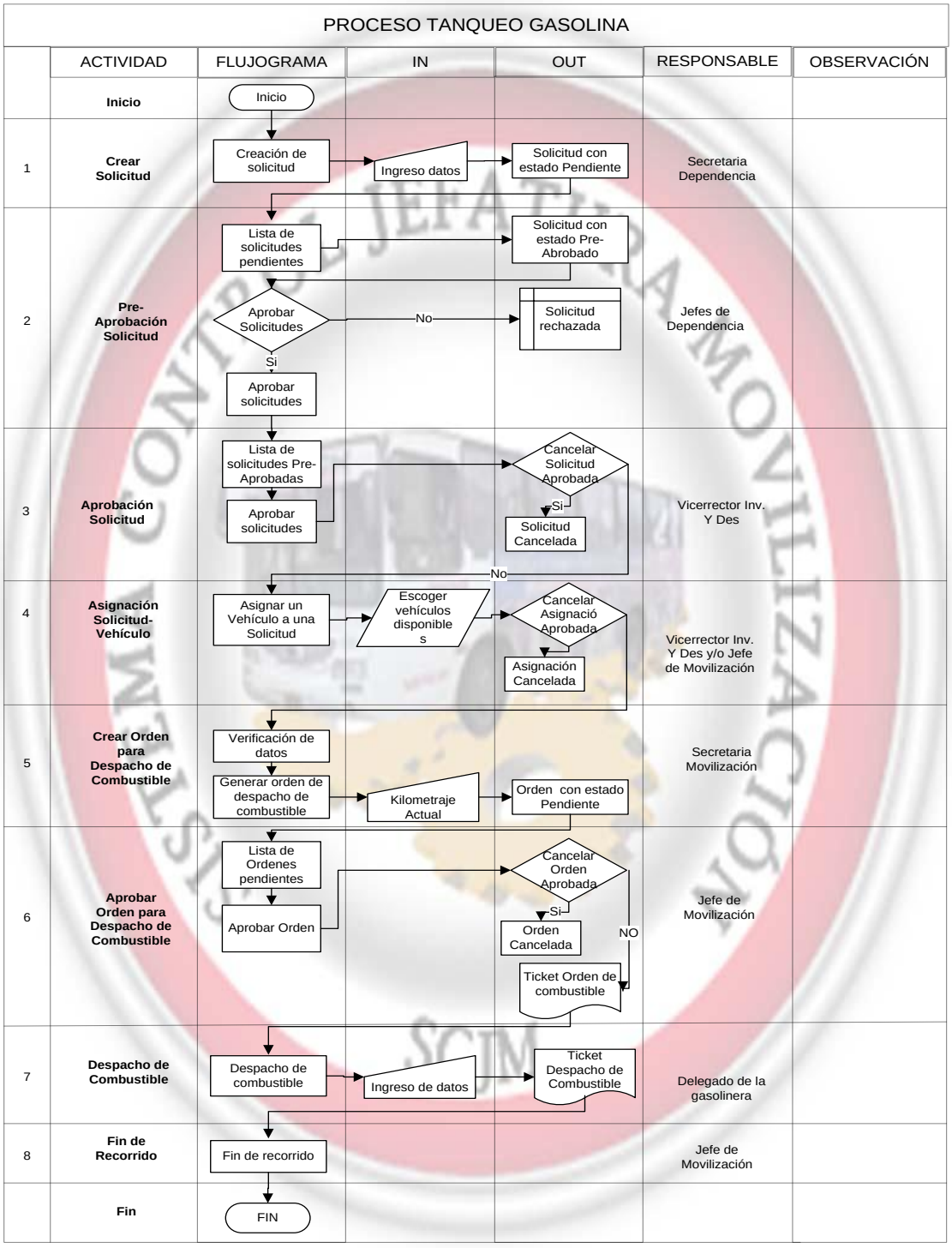


Figura IV. 39. Diagrama de proceso de Tanqueo

Tambien podemos destacar el cronograma de actividades en la figura 40:

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	Recolección de Datos	5 días	lun 18/06/12	vie 22/06/12	
2	**Gestión del Proyecto**	**5 días**	**lun 25/06/12**	**vie 29/06/12**	**1**
3	Planificación Inicial	1 día	lun 25/06/12	lun 25/06/12	
4	Iteración 1	1 día	mar 26/06/12	mar 26/06/12	3
5	Iteración 2	1 día	mié 27/06/12	mié 27/06/12	4
6	Iteración 3	1 día	jue 28/06/12	jue 28/06/12	5
7	Diario de Actividades	1 día	vie 29/06/12	vie 29/06/12	6
8	Implementación	174 días	lun 02/07/12	jue 28/02/13	2
9	Pruebas	23 días	vie 01/03/13	mar 02/04/13	8

Figura IV. 40. Cronograma de Actividades

4.2.2. Implementación

4.2.2.1. Base de datos

Nuestra base de datos consta de tres esquemas:

- Master: En este esquema se maneja todo lo concerniente con la seguridad, creación de menús y todo lo que tiene que ver con las personas, dependencias y cargos.
- Orden: en este esquema se encuentran las tablas involucradas en el proceso principal.
- Vehículo-chofer: en este esquema se encuentra todo lo referente al vehículo con las tablas de la características del vehículo así como las del soat, matricula y seguro; también podemos encontrar todo lo referente al conductor con las licencia

El diseño de los esquemas de la Base de Datos podrá ser apreciado de mejor manera en el Anexo 3.

4.2.2.2.Prototipos interfaces de usuario finales

Con la descripción detallada de las historias de usuario y con los diagramas de procesos podemos definir las interfaces de usuario finales (Ver figura 41 a la figura 45) las cuales serán implantadas en el sistema.

Figura IV. 41. Control de Acceso de Usuarios

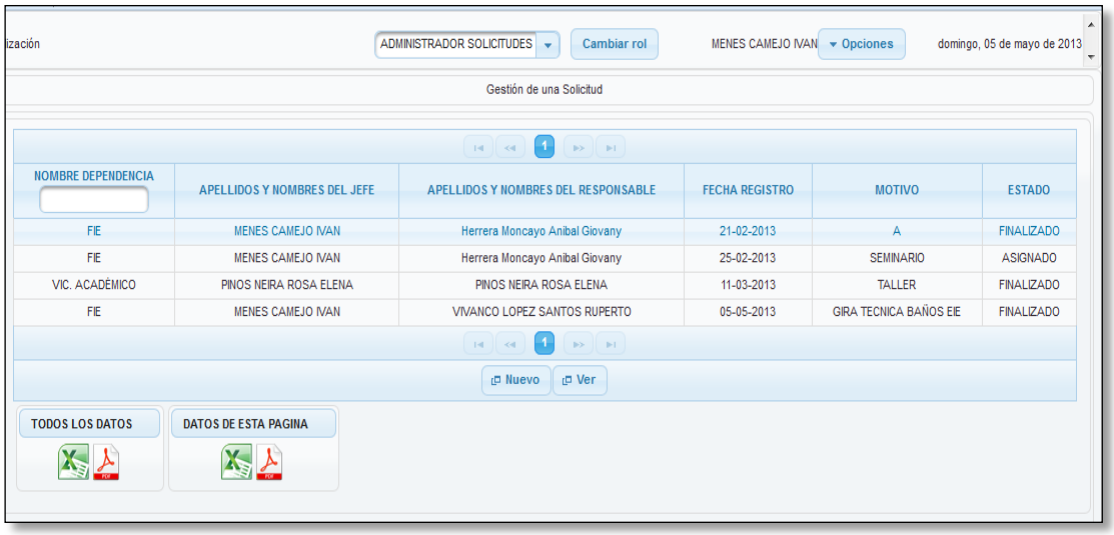


Figura IV. 42. Gestión de solicitud

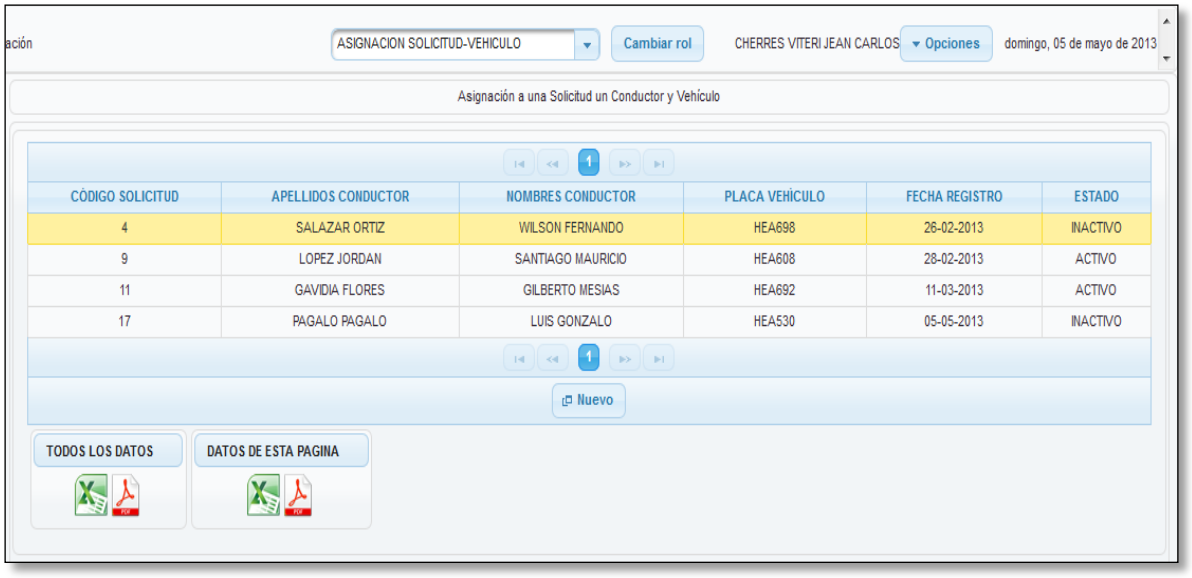


Figura IV. 43. Gestión Asignación Solicitud-Vehículo

Figura IV. 44. Gestión de Orden Combustible

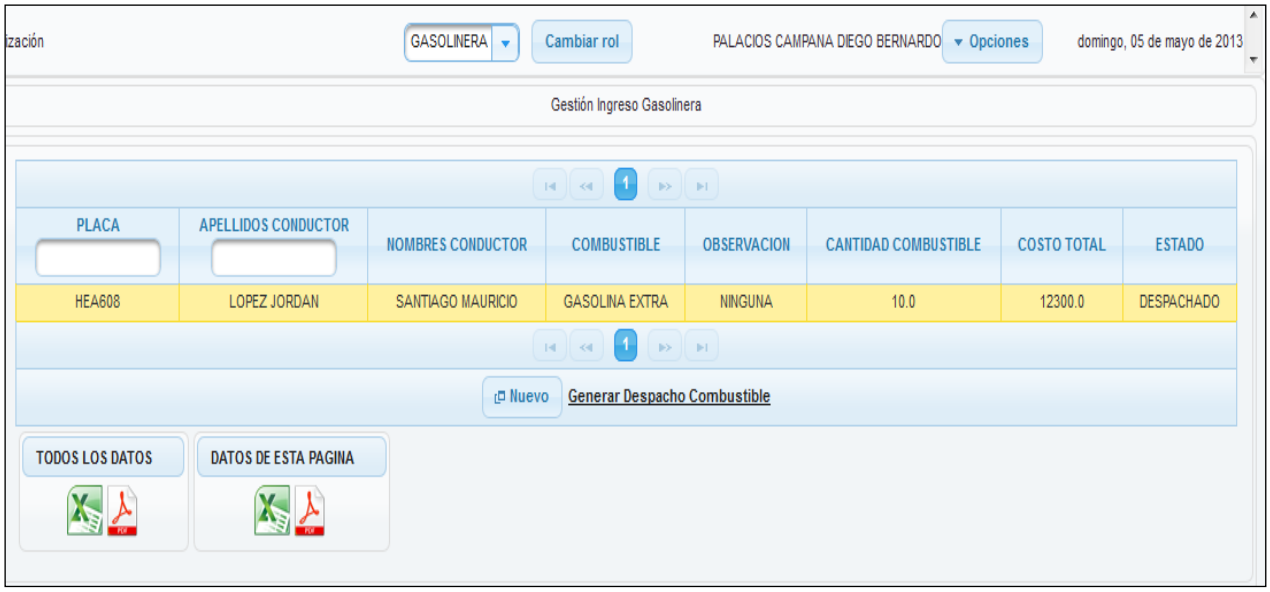


Figura IV. 45. Gestión Ingreso Gasolinera

4.2.2.3.Código fuente

El código fuente lo hemos tomado de la solicitud, puesto que todas y cada una de las clases, funciones controladores y páginas xhtml son similares en estructura. Ver Anexo 3.

4.2.3. Pruebas

Las pruebas se convierten en una herramienta de desarrollo, no un paso de verificación que puede despreciarse si a uno le parece que el código está bien. Las pruebas son creadas a partir de las historias de usuario. Durante una iteración la historia de usuario seleccionada en la planificación de iteraciones se convertirá en una prueba de aceptación. El usuario especifica los aspectos a testear cuando una historia de usuario ha sido correctamente implementada. Una historia de usuario puede tener más de una prueba de aceptación, tantas como sean necesarias para garantizar su correcto funcionamiento.

Cada una de ellas representa una salida esperada del sistema. Una historia de usuario no se considera completa hasta que no supera sus pruebas de aceptación.

A continuación se presenta las pruebas realizadas a cada una de las historias de usuarios del sistema y cada una de estas consta con su tabla de pruebas respectivamente.

Historia 1

Tabla IV. XXXV. Pruebas. Historia 1.

Fecha	Descripción	Autor
01/03/2013	Pruebas y Modificaciones	Karla Sosa / Eulalia Carrillo

Control de Acceso a Usuarios

Descripción

Hay seis tipos de usuarios: Administrador, Vicerrector, Jefe de Dependencia, Jefe de Movilización, Conductor, Delegado de Gasolinera.

Condiciones de ejecución

Cada uno de los usuarios deben constar en la base de datos con roles y permisos de acceso a los menús dependiendo de las funciones que les corresponden.

Entrada

- El usuario introducirá su cuenta y clave.
- El proceso de control de Acceso a Usuarios finaliza.

Resultado esperado

Tras ingresar el usuario su cuenta y clave respectiva, debe aparecer automáticamente el menú principal para cualquier caso de los tipos de usuarios.

Evaluación de la prueba

Prueba satisfactoria.

Historia 2

Tabla IV. XXXVI Pruebas. Historia 2.

Fecha	Descripción	Autor
04/03/2013	Pruebas y Modificaciones	Karla Sosa / Eulalia Carrillo

Control de conductores

Descripción

El Jefe de Movilización podrá ingresar, modificar un nuevo conductor con su respectiva licencia.

Condiciones de ejecución

El Jefe de Movilización debe constar en la base de datos del Sistema

Entrada

- El Jefe de Movilización una vez que ingresa al sistema, escoge el rol ADMINISTADOR MOVILIZACIÓN
- Escoge el submenú GESTIÓN CONDUCTOR
- Se ingresara los datos tanto del conductor como de su licencia
- Se puede ingresar los tipos de licencia.

Resultado esperado

Tras ingresar el Jefe de Movilización los datos de un nuevo conductor con su licencia se insertan exitosamente. De la mima manera los tipos de licencia.

Evaluación de la prueba

Prueba satisfactoria.

Historia 3

Tabla IV. XXXVII. Pruebas. Historia 3.

Fecha	Descripción	Autor
06/03/2013	Pruebas y Modificaciones	Karla Sosa / Eulalia Carrillo

Control de vehículos

Descripción

El Jefe de Movilización podrán ingresar, modificar un nuevo vehículo con su Soat, Seguro y Matricula correspondiente.

Condiciones de ejecución

Los usuarios y algunos datos de los vehículos deben tener las Marcas, Modelos, Colores, Clases, Soat, Seguro y Matriculas previamente registradas en la base de datos.

Entrada

- El Jefe de Movilización una vez que ingresa al sistema, escoge el rol ADMINISTADOR MOVILIZACIÓN VEHICULOS
- Escoge el submenú NUEVO
- El usuario insertara todos los datos del vehículo para ser registrados.
- En caso de no existir una de las Marcas, Modelos, Colores, Clases, Soat, Seguro y Matriculas se debe primero proceder al ingreso del dato inexistente.

Resultado esperado

Tras ingresar el Jefe de Movilización los datos de un nuevo vehículo se insertan exitosamente en la base de datos.

Evaluación de la prueba

Prueba satisfactoria.

Historia 4

Tabla IV. XXXVIII. Pruebas. Historia 4.

Fecha	Descripción	Autor
07/03/2013	Pruebas y Modificaciones	Karla Sosa / Eulalia Carrillo

Control de recorridos

Descripción

Jefe de Movilización y/o Conductor deben autenticarse en el sistema y dependiendo de su rol deberán realizar las actividades que les corresponden

Condiciones de ejecución

Los usuarios así como las rutas deben constar en la base de Datos de la aplicación.

Entrada

- El Jefe de Movilización una vez que ingresa al sistema, escoge el rol ADMINISTADOR MOVILIZACION
- Escoge el submenú GESTION RUTAS DIARIAS
- Se podrá visualizar todas las rutas realizadas por los conductores.
- En el caso del conductor una vez que ingresa al sistema, ya tiene definido su rol y podrá escoger la opción Ver mis Datos y Nueva Ruta
- Ingreso de una nueva ruta

Resultado esperado

Tras ingresar el Jefe de Movilización puede acceder a los datos de los recorridos de los conductores en general y el Conductor al ingresar un nuevo recorrido se insertan exitosamente en la base de datos

Evaluación de la prueba

Prueba satisfactoria.

Historia 5

Tabla IV. XXXIX. Pruebas. Historia 5.

Fecha	Descripción	Autor
11/03/2013	Pruebas y Modificaciones	Karla Sosa / Eulalia Carrillo

Control de la Seguridad

Descripción

El Administrador del sistema una vez que ingresa al sistema tiene el control total sobre el mismo, en cuanto tiene que ver a la seguridad podrá realizar diferentes actividades dependiendo de la necesidad que se presente.

Condiciones de ejecución

El administrador debe constar en la base de Datos de la aplicación.

Entrada

- El Administrador del sistema una vez que ingresa al sistema, aparece el rol ADMINISTRADOR MASTER
- Escoge los submenús ADMINISTRACION USUARIOS, dependiendo de la necesidad escogerá Gestión de cuentas, Gestión de roles al Sistema y Asignación de roles a la cuenta insertando uno nuevo por cada uno de ellos dependiendo el caso
- Escoge los submenús ADMINISTACION MODULOS, dependiendo de la necesidad escogerá Gestión Modulo y Asignar Grupo Menú Cero Al Modulo este submenú sirve para crear menús ya que son dinámicos
- Escoge los submenús ADMINISTACION PERSONAS YDEPENDENCIAS dependiendo de la necesidad escogerá Gestión Dependencia, Asignar Persona a una Dependencia con su Cargo y Migrar Personas insertando uno nuevo por cada uno de ellos dependiendo el caso

Resultado esperado

Tras ingresar el administrador dependiendo la actividad que realice se insertaron los datos exitosamente.

Evaluación de la prueba

Prueba satisfactoria.

Historia 6

Tabla IV. XL. Pruebas. Historia 6.

Fecha	Descripción	Autor
13/03/2013	Pruebas y Modificaciones	Karla Sosa / Eulalia Carrillo

Control Conductor-Dependencia-Vehículo

Descripción

Jefe de Movilización debe autenticarse en el sistema y deberá realizar las actividades que les corresponden

Condiciones de ejecución

El Jefe de Movilización así como un conductor a una dependencia con su conductor, estos ya deberán estar ingresados en la base de datos.

Entrada

- El usuario una vez ingresa al sistema, escoge el rol ADMINISTRADOR MOVILIZACION VEHICULOS
- Escoger el Submenú Gestión Conductor-Dependencia-Vehículo
- Ingreso de un nuevo Conductor-Dependencia-Vehículo

Resultado esperado

Tras ingresar el Jefe de Movilización puede acceder a los datos Conductor-Dependencia-Vehículo los cuales se insertan exitosamente en la base de datos

Evaluación de la prueba

Prueba satisfactoria.

Historia 7

Tabla IV. XLI. Pruebas. Historia 7.

Fecha	Descripción	Autor
15/03/2013	Pruebas y Modificaciones	Karla Sosa / Eulalia Carrillo

Control solicitud

Descripción

Jefe de Dependencia debe autenticarse en el sistema y deberá realizar las actividades que les corresponden

Condiciones de ejecución

El Jefe de Dependencia, jefe y responsable debe constar en la base de Datos de la aplicación.

Entrada

- El Jefe de Dependencia vez registrado en el sistema, ésta ya tiene definido su rol.
- Escoger la opción Creación Solicitud Movilización.
- Insertar una nueva Solicitud.
- Jefe de Dependencia dará el visto bueno y enviará la solicitud, y el Vicerrector deberá aprobar las solicitudes que se creen.

Resultado esperado

Tras ingresar el Jefe Dependencia puede acceder a los datos de la Solicitud los cuales se insertan exitosamente en la base de datos

Evaluación de la prueba

Prueba satisfactoria.

Historia 8

Tabla IV. XLII. Pruebas. Historia 8.

Fecha	Descripción	Autor
19/03/2013	Pruebas y Modificaciones	Karla Sosa / Eulalia Carrillo

Control Asignación

Descripción

Jefe de Movilización y/o Vicerrector debe autenticarse en el sistema y deberá realizar las actividades que les corresponden

Condiciones de ejecución

Jefe de Movilización y Vicerrector así como las solicitudes aprobadas deben constar en la base de Datos de la aplicación.

Entrada

- Jefe de Movilización y/o Vicerrector una vez registrado en el sistema, ésta ya tiene definido su rol ASIGNACION SOLICITUD-VEHICULO
- Escoger asignar Solicitud-Vehículo
- Insertar una asignación a una solicitud determinada el vehículo con la cual se realizara dicha solicitud.

Resultado esperado

Tras ingresar el Jefe de Movilización y/o Vicerrector puede acceder a los datos de la Asignación Solicitud-Vehículo los cuales se insertan exitosamente en la base de datos

Evaluación de la prueba

Prueba satisfactoria.

Historia 9

Tabla IV. XLIII. Pruebas. Historia 9.

Fecha	Descripción	Autor
21/03/2013	Pruebas y Modificaciones	Karla Sosa / Eulalia Carrillo

Control orden combustible

Descripción

Jefe de Movilización debe autenticarse en el sistema y deberá realizar las actividades que les corresponden

Condiciones de ejecución

Jefe de Movilización así como las asignaciones aprobadas deben constar en la base de Datos de la aplicación.

Entrada

- Movilización una vez registrado en el sistema, deberá escoger el rol ADMINISTRADOR MOVILIZACION.
- Escoger el menú Gestión Tanqueo
- Escoger una orden para Despacho de combustible
- Insertar una nueva orden para Despacho de combustible.

Resultado esperado

Tras ingresar el Jefe de Movilización puede acceder a los datos de una orden para Despacho de combustible los cuales se insertan exitosamente en la base de datos

Evaluación de la prueba

Prueba satisfactoria.

Historia 10

Tabla IV. XLIV. Pruebas. Historia 10.

Fecha	Descripción	Autor
25/03/2013	Pruebas y Modificaciones	Karla Sosa / Eulalia Carrillo

Control ingreso gasolinera

Descripción

El Delegado de la Gasolinera debe autenticarse en el sistema y deberá realizar las actividades que les corresponden

Condiciones de ejecución

El Delegado de la Gasolinera así como las órdenes aprobadas deben constar en la base de datos de la aplicación.

Entrada

- El Delegado de la Gasolinera una vez registrado en el sistema, éste ya tiene definido su rol Gasolinera.
- Escoger la opción Despacho.
- Escoger la opción Despacho de Combustible.
- Insertar un nuevo Despacho de Combustible.

Resultado esperado

Tras ingresar el Delegado de la Gasolinera puede acceder a los datos de un Despacho de combustible los cuales se insertan exitosamente en la base de datos

Evaluación de la prueba

Prueba satisfactoria.

Historia 11

Tabla IV. XLV. Pruebas. Historia 11.

Fecha	Descripción	Autor
27/03/2013	Pruebas y Modificaciones	Karla Sosa / Eulalia Carrillo

Término de Proceso

Descripción

El Jefe de Movilización debe autenticarse en el sistema y deberá realizar las actividades que les corresponden

Condiciones de ejecución

El Jefe de Movilización así como los ingresos en la Gasolinera aprobadas deben constar en la base de datos de la aplicación.

Entrada

- El Jefe de Movilización una vez registrado en el sistema, éste deberá escoger el rol ADMINISTRADOR MOVILIZACION
- Escoger el menú Gestión Tanqueo
- Escoger el vinculo Entrega Vehículo

Resultado esperado

Tras ingresar el Jefe de Movilización puede acceder a los datos de un Entrega Vehículo los cuales se insertan exitosamente en la base de datos

Evaluación de la prueba

Prueba satisfactoria.

Historia 12

Tabla IV. XLVI. Pruebas. Historia 12.

Fecha	Descripción	Autor
27/03/2013	Pruebas y Modificaciones	Karla Sosa / Eulalia Carrillo

Emisión de reportes

Descripción

El Jefe de Movilización debe autenticarse en el sistema y deberá realizar las actividades que les corresponden

Condiciones de ejecución

El Jefe de Movilización debe constar en la base de datos de la aplicación.

Entrada

- El Jefe de Movilización una vez registrado en el sistema, éste deberá escoger el rol ADMINISTRADOR MOVILIZACION
- Escoger el menú Reportes
- Escoger el reporte que necesite

Resultado esperado

Tras ingresar el Jefe de Movilización puede acceder a los datos de los reportes exitosamente en la base de datos

Evaluación de la prueba

Prueba satisfactoria.

Historia 13

Tabla IV. XLVII. Pruebas. Historia 13.

Fecha	Descripción	Autor
02/04/2013	Pruebas y Modificaciones	Karla Sosa / Eulalia Carrillo

Emisión de tickets para la orden y el ingreso de gasolinera

Descripción

El Jefe de Movilización y Delegado de Gasolinera debe autenticarse en el sistema y deberá realizar las actividades que les corresponden

Condiciones de ejecución

El Jefe de Movilización y Delegado de Gasolinera así como las órdenes aprobadas deben constar en la base de datos de la aplicación.

Entrada

- El Jefe de Movilización y Delegado de Gasolinera una vez registrado en el sistema, cada uno con su rol respectivamente.
- Generar las Ordenes de Combustible se deberá imprimir un ticket de constancia así como en la gasolinera de un ticket de despacho de combustible

Resultado esperado

Tras ingresar el Jefe de Movilización y Delegado de Gasolinera puede acceder a los datos de los reportes con los tickets exitosamente en la base de datos

Evaluación de la prueba

Prueba satisfactoria.

CONCLUSIONES

1. Para evaluar las librerías de componentes que fueron escogidas como objeto de estudio en la presente investigación se tomó en cuenta los parámetros Librerías más Actuales y Librerías con Mayor Madurez, definidos cuidadosa y subjetivamente.

2. Se seleccionó las tres librerías más adecuadas para el desarrollo de aplicaciones web enriquecidas, Richfaces, Icefaces y Primefaces.

3. Se llevó a cabo la creación de prototipos que permitieron realizar un análisis comparativo más detallado determinando que Primeface con un 46.15% es la librería que tiene un tamaño de página y respuesta Ajax menor y por ende la librería más adecuada.

4. A partir de la selección de la librería más adecuada para el desarrollo de aplicaciones web enriquecidas, Primefaces, se desarrollo el Sistema de Control Vehicular de la ESPOCH.

RECOMENDACIONES

1. Estudiar, analizar y comparar librerías de componentes debe ser un proceso sistemático y organizado, ya que de este proceso depende la elección de la librería más adecuada para el desarrollo de aplicaciones web enriquecidas.

2. Tomar en cuenta cuidadosamente los parámetros definidos para obtener una selección de librerías de componentes más concreta.

3. Aplicar el uso de la librería de componentes con menor tamaño de página y de respuesta Ajax, como lo es Primefaces ayudará para el desarrollo de aplicaciones web enriquecidas con JSF.

4. Se recomienda al Administrador del Sistema SCJM que en el caso de existir nuevos usuarios, sean agregados tomando en cuenta los permisos que se le asigna para que el tratamiento de la información sea la más idónea.

RESUMEN

"Estudio comparativo de librerías de componentes para desarrollo de aplicaciones web con interfaces enriquecidas con JSF, aplicado al Sistema de Control Vehicular de la ESPOCH"
El Método Inductivo se usó para el análisis e implementación del proyecto permitiendo investigar requerimientos, código, componentes, recursos, instrumentos necesarios para la implementación del sistema. El Método Deductivo fue útil para tratar temas generales analizados de tal forma que permitió tener una percepción global del Sistema de Control Vehicular SCJM.

Los materiales usados para el desarrollo del SCJM fueron: Sistema Operativo Windows7, motor de Base de Datos Postgres 9.1, IDE Netbeans 7.1, dos laptops, servidores facilitados por la ESPOCH tales como Servidor de Control de Versiones-Subversión con Windows server 2003 Enterprise Edition Sp2, Servidor de Base Datos Postgres 9.1 con Centos 5.0, Servidor Web Glassfish Server Open SourceEdition 3.1.1 Build 12 con Centos 5.0 y la herramienta DotProject para el control de tareas.

Se realizó el estudio, análisis, comparación de librerías de componentes y su respectiva interpretación de valores cuantitativos obteniéndose los siguientes puntajes: Primefaces con 46.15%, Icefaces con el 23.08% y Richfaces con el 30.77%. Se determinó así que la librería más adecuada es Primefaces por su calidad de componentes, por disponer de una buena documentación, por su compatibilidad con otros navegadores, por su facilidad de uso.

Se concluye, que al elaborar el Sistema de Control Vehicular de la ESPOCH, desarrollado en el DESITEL, la información del Departamento de Movilización tendrá un manejo, administración y control adecuados.

Recomendamos para el correcto uso del sistema se apliquen los manuales técnicos y de usuario del SCJM y con ello el Departamento de Movilización tenga un excelente desenvolvimiento a nivel institucional.

SUMARY

Comparative research of component libraries for developing web applications with interfaces enriched with JSF, applied to the Vehicle Control System of ESPOCH. the inductive method was used for the analysis and implementation of the project allowing for investigation of requirements, code, components, resources, tools required for system implementation. The Deductive method was useful for treating general topics analyzed a way that allowed having an overall perception of vehicle control system.

The materials used for the development of SCJM were: operating system windows 7 Postgres 9.1 database engine, IDE Netbeans 7.1, two ¡laptops, servers provided by the ESPOCH such as servant: of control of versions - subversions with Windows 2003 Enterprise Edition Sp2, Postgres 9.1 Database server with Centos 5.0, Web Glassfish Server Open Source Edition 3.1.1 Build 12 with Centos 5.0 and DotProject tool for the control of tasks.

The research was established, the analysis, comparison of components libraries and their respective interpretation of quantitative values being obtained; the following scores: Primefaces with 46.15%, Icefaces with 23.08% and Richfaces with 30.77%. It was determined that the most appropriate library is Primefaces for its quality of component it has good documentation, for its compatibility with other browsers, for its facility of use

It was concluded that in developing the Vehicle Control System of ESPOCH, developed a DESITEL information of Mobilization Department will have a management, administration and control.

We recommend to the proper use of the system are implemented technical: and use manuals of SCJM and with it the Mobilization Department has an excellent development at the institutional level.

GLOSARIO

COMPONENTE: Es una clase abstracta que representa todo lo que tiene una posición, un tamaño, puede ser pintado en pantalla y puede recibir eventos.

LIBRERÍA DE COMPONENTES: Componentes agrupados con o sin criterio pero lo ideal es que se dediquen a funciones específicas

AJAX: es una tecnología asíncrona, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página.

VALIDACIÓN: proceso de revisión al que se somete un programa informático para comprobar que cumple con sus especificaciones. Dicho proceso, que suele tener lugar al final de la etapa de desarrollo, se realiza principalmente con la intención de confirmar que el software esté en condiciones de desarrollar las tareas que el **usuario** que lo adquiere planea llevar a cabo.

APLICACIÓN: es un tipo de programa informático diseñado como herramienta para permitir a un usuario realizar uno o diversos tipos de trabajos.

KIT RENDERIZADOR: define como se mapean las clases de los componentes a las etiquetas de componentes apropiadas para un cliente particular.

ETIQUETA: también conocida como *TAG*. Es una marca con clase que delimita una región en los lenguajes basados en XML.

BEAN: es un componente software que tiene la particularidad de ser reutilizable y así evitar la tediosa tarea de programar los distintos componentes uno a uno. Un Bean puede representar desde un botón, un grid de resultados, un panel contenedor o un simple campo de texto, hasta otras soluciones mucho más complejas como conexiones a bases de datos, etc.

CONTENEDORES AJAX. *AjaxContainer* es una interfaz que describe un área de una página JSF que debe ser decodificado en una petición Ajax.

PROTOTIPO: representación limitada de un producto, permite a las partes probarlo en situaciones reales o explorar su uso, creando así unproceso de diseño de iteración que genera calidad.

HISTORIAS DE USUARIOS: es una representación de un requisito de software escrito en una o dos frases utilizando el lenguaje común del usuario. Las historias de usuario son utilizadas en las metodologías de desarrollo ágiles para la especificación de requisitos

ITERACIÓN: significa el acto de repetir un proceso con el objetivo de alcanzar una meta deseada, objetivo o resultado. Es la repetición de un proceso dentro de un programa.

INCIDENCIA: Circunstancia o sucesos secundarios que ocurren durante el desarrollo, pero que puede influir en el resultado final.

FLUJO DE PROCESOS: descripción visual de las actividades implicadas en un proceso mostrando la relación secuencial ente ellas, facilitando la rápida comprensión de cada actividad y su relación con las demás.

ANEXOS

Anexo 1. Código Fuente de creación de xhtml de prueba con la tabla color_vehiculo, diseño de la paginación.

Richfaces

```
<f:view xmlns="http://www.w3.org/1999/xhtml"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:rich="http://richfaces.org/rich">
   <rich:dataScroller for="colorList" maxPages="15" reRender="colorList"/>
   <rich:dataTable id="colorList" rows="5" value="#{colorBean.colorVehiculo}"
var="color">
     <rich:column>
         <f:facet name="header">
            <h:outputText value="Código"/>
         </f:facet>
         <h:outputText value="#{color.codigo}"/>
      </rich:column>
      <rich:column>
         <f:facet name="header">
            <h:outputText value="Codificación"/>
         </f:facet>
        <h:outputText value="#{color.codificacion}"/>
```

```
        </rich:column>
        <rich:column>
           <f:facet name="header">
              <h:outputText value="Nombre"/>
           </f:facet>
           <h:outputText value="#{color.nombre}"/>
        </rich:column>
    </rich:dataTable>
</f:view>
```

Icefaces

```
<f:view xmlns=”http://www.w3.org/1999/xhtml” xmlns:f=”http://java.sun.com/jsf/core”
    xmlns:ice=”http://www.icesoft.com/icefaces/component”
xmlns:h=”http://java.sun.com/jsf/html”>
    <ice:panelGroup>
       <ice:dataPaginator for=”data” paginator=”true” fastStep=”10″
paginatorMaxPages=”10″>
           <f:facet name=”first”>
              <ice:graphicImage url=”/xmlhttp/css/rime/css-images/arrow-first.gif”/>
           </f:facet>
           <f:facet name=”last”>
              <ice:graphicImage url=”/xmlhttp/css/rime/css-images/arrow-last.gif”/>
           </f:facet>
```

```
        <f:facet name="previous">
            <ice:graphicImage url="/xmlhttp/css/rime/css-images/arrow-previous.gif"/>
        </f:facet>
        <f:facet name="next">
            <ice:graphicImage url="/xmlhttp/css/rime/css-images/arrow-next.gif"/>
        </f:facet>
        <f:facet name="fastforward">
            <ice:graphicImage url="/xmlhttp/css/rime/css-images/arrow-ff.gif"/>
        </f:facet>
        <f:facet name="fastrewind">
            <ice:graphicImage url="/xmlhttp/css/rime/css-images/arrow-fr.gif"/>
        </f:facet>
    </ice:dataPaginator>
</ice:panelGroup>
<ice:panelGroup>
    <ice:dataTable id="data" var="book" value="#{dataTableBean.books}" rows="5″>
        <ice:column>
            <f:facet name="header">
                <h:outputText value="isbn"/>
            </f:facet>
            <h:outputText value="#{book.isbn}"/>
        </ice:column>
        <ice:column>
```

```
            <f:facet name=”header”>
                <h:outputText value=”author”/>
            </f:facet>
            <h:outputText value=”#{book.author}”/>
          </ice:column>
          <ice:column>
            <f:facet name=”header”>
                <h:outputText value=”title”/>
            </f:facet>
            <h:outputText value=”#{book.title}”/>
          </ice:column>
        </ice:dataTable>
    </ice:panelGroup>
</f:view>
```

Primefaces

```
<f: xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.org/ui"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:f="http://java.sun.com/jsf/core">
    <p:dataTable var=”color” value=”#{colorVehiculoC.colores}” paginator=”true”
rows=”5″ paginatorPosition=”top” paginatorTemplate=”{CurrentPageReport}
```

```
{FirstPageLink} {PreviousPageLink} {PageLinks} {NextPageLink} {LastPageLink}
{RowsPerPageDropdown}"  rowsPerPageTemplate="5,10,15″>
      <p:column>
         <f:facet name="header">
            <h:outputText value="Código"/>
         </f:facet>
         <h:outputText value="#{color.codigo}"/>
      </p:column>
      <p:column>
         <f:facet name="header">
            <h:outputText value="Codificación"/>
         </f:facet>
         <h:outputText value="#{ color.codificacion}"/>
      </p:column>
      <p:column>
         <f:facet name="header">
            <h:outputText value="Nombre"/>
         </f:facet>
         <h:outputText value="#{ color.nombre}"/>
      </p:column>
   </p:dataTable>
</f:view>
```

Anexo 2. Herramienta NeoLoad

Una vez creados los prototipos se procederá a realizar las pruebas a cada uno de ellos utilizando la herramienta NeoLoad, de la cual se describirá la forma de uso y los resultados obtenidos.

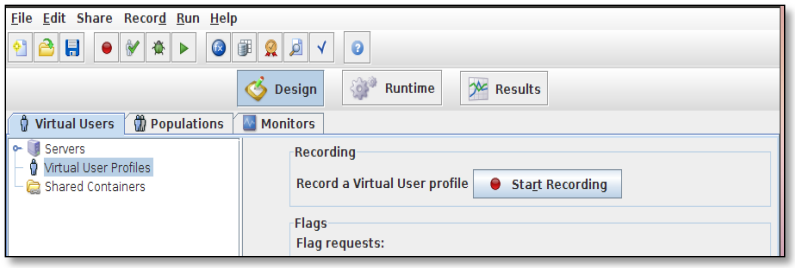
Splash herramienta NeoLoad

Para realizar la medida de la longitud de la página y respuestas AJAX debemos como primer paso realizar la captura del prototipo que vamos a medir, nos referimos a grabar todo el proceso que hará el prototipo, en este caso la herramienta grabará los pasos que realizamos desde el ingreso al prototipo hasta cuando finalicemos la sesión de trabajo, en este caso para la medición de los prototipos se especificara los pasos de uno de ellos ya que el procedimiento es similar.

1.- Ingresamos a la herramienta y lo primero que observaremos es el menú de Proyectos, seleccionamos **New Project**

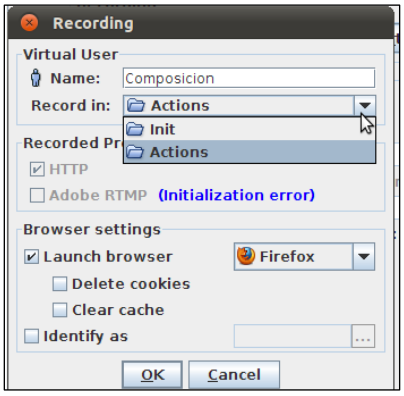
Opciones de Herramienta NeoLoad

Una vez cargada la herramienta, seleccionamos la opción **StartRecording:**



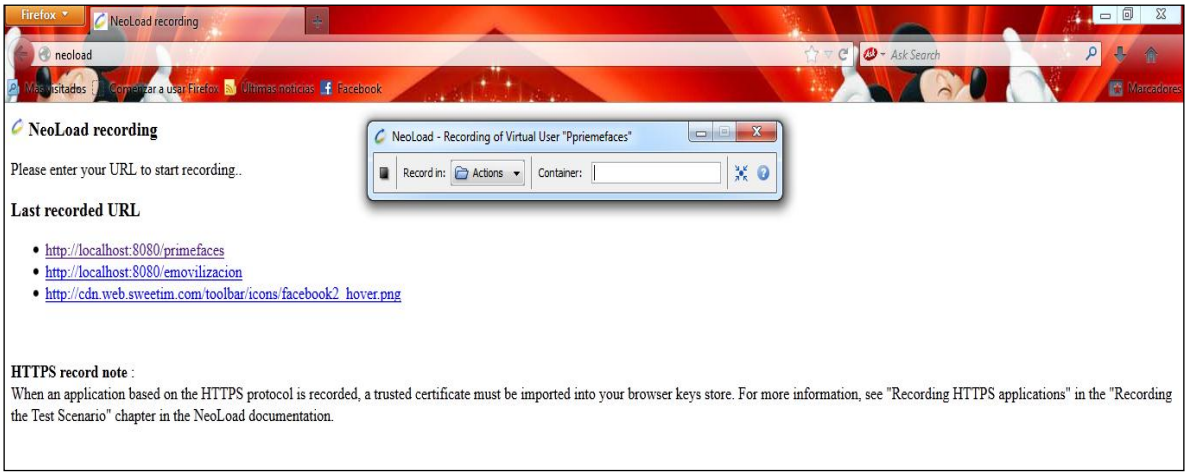
Botón de inicio de Grabación

Aparecerá una ventana donde especificaremos el nombre de nuestro entorno virtual y en **Record in** seleccionaremos **Init** que se refiere a la parte inicial del sistema que es por donde se empieza la captura, este tipo de diferenciaciones la herramienta lo utiliza para clasificar cada parte del sistema, así por ejemplo la gestión de los datos del usuario y del proyecto serán almacenados en **Actions**. Presionamos **OK.**



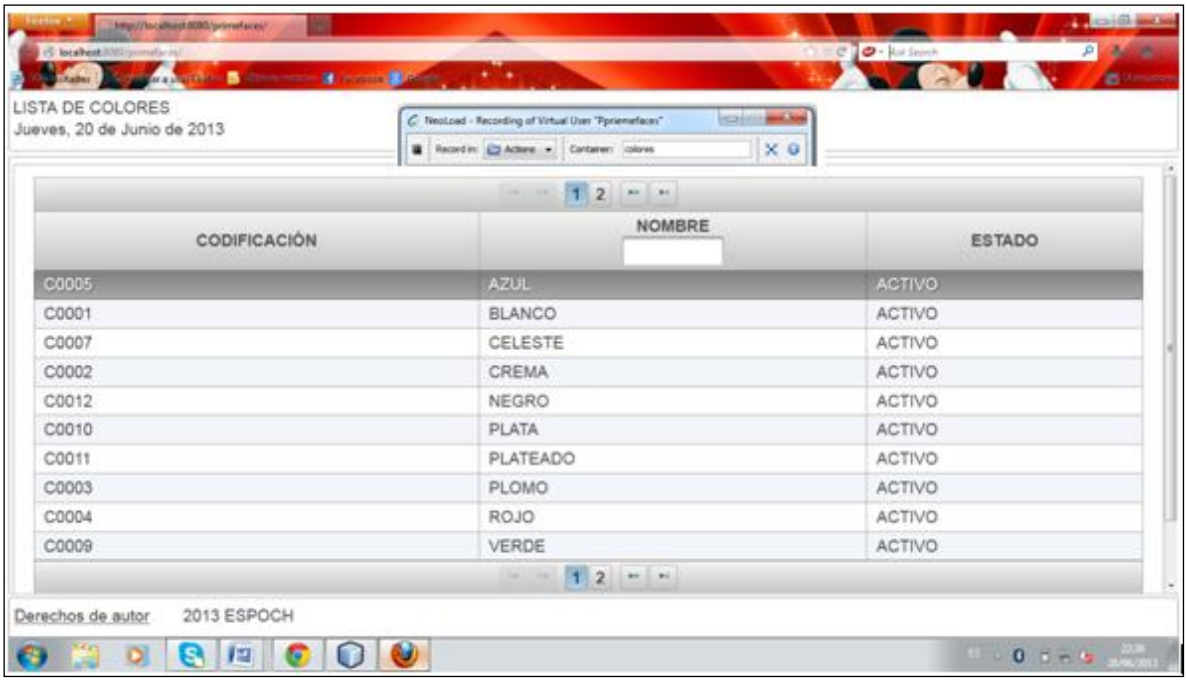
Ventana de Propiedades de la grabación

Al finalizar la Configuración se abrirá nuestro navegador predeterminado, el cual nos indicará todas las direcciones que hayamos abierto recientemente y pulsamos sobre el enlace que deseemos grabar:



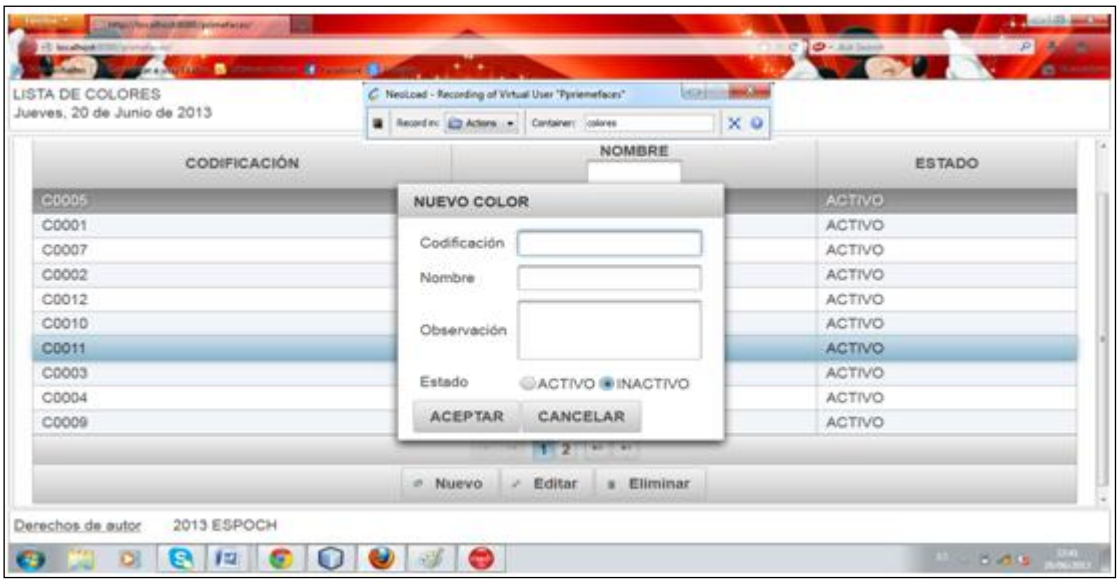
Selección de enlace a grabar

En la parte superior central de la página observaremos que la herramienta se encuentra grabando desde el inicio, en la casilla **Container** se indicará un nombre que nos servirá para poder identificar que parte del sistema se encuentra en funcionamiento.



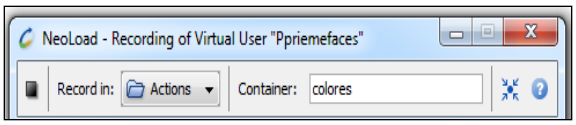
Selección de enlace a grabar

Al momento de realizar la gestión nuevo, editar y eliminar debemos especificar que parte estamos grabando, ingresaremos un nombre que identifique el proceso a realizar. De igual forma especificaremos el proceso referente a proyectos.



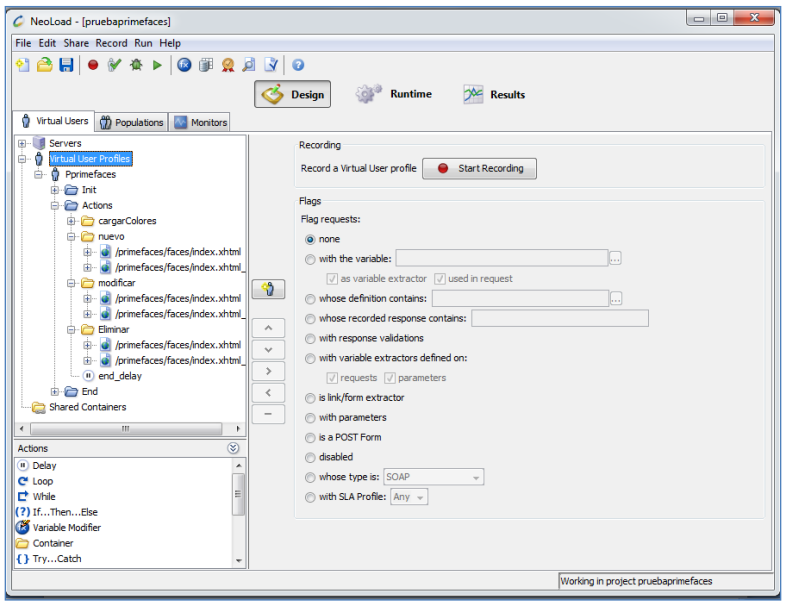
Grabación de nuevo

Una vez hayamos terminado de grabar los procesos procederemos a cerrar la grabación haciendo click en el botón **Stop** que se encuentra en la ventana superior central.



Botón de alto a grabación del sistema

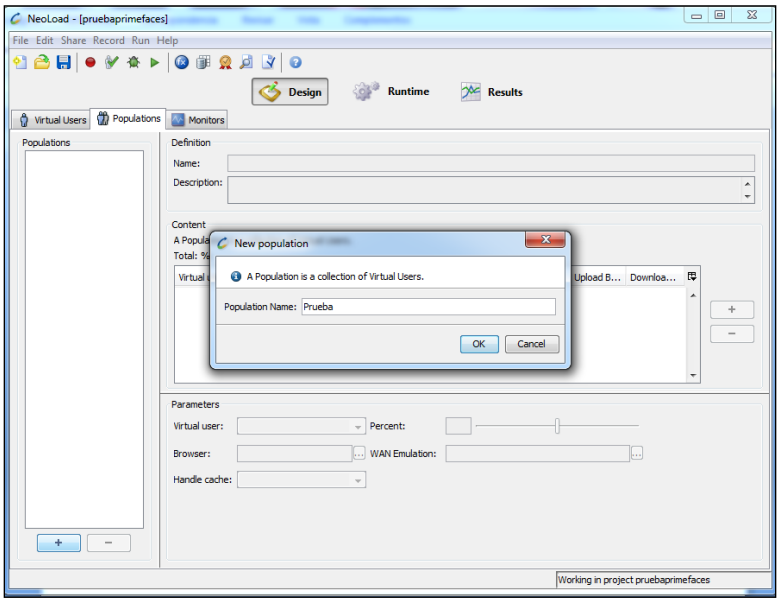
Una vez detenida la grabación esperaremos a que termine la creación del entorno virtual que se empieza a ejecutar una vez hayamos cerrado la grabación, al finalizar este paso se nos mostrará en la ventana que aparece a continuación los pasos que grabamos separados de acuerdo a la posición como hayamos decidido ubicarlos, se puede apreciar que se verán las páginas del sistema por las que navegamos o que tomaron parte durante la grabación.



Pasos capturados del Prototipo

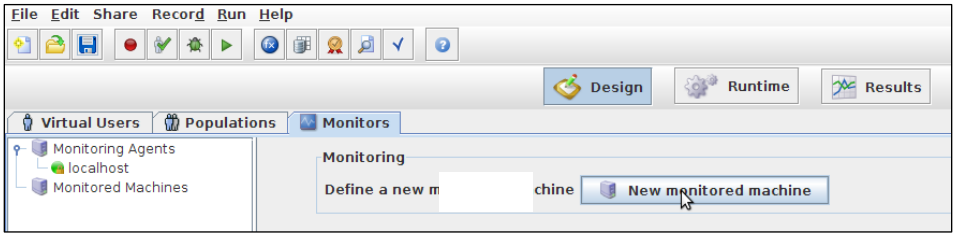
A continuación se procedera a la creación de población, es decir el número de usuarios virtuales para utilizar el sistema durante la ejecución de la simulación de uso. Para ellos seleccionamos la pestaña **Populations,** hacemos click en el botón + ubicado en la parte inferior izquierda de la ventana, aparecerá una ventana donde indicaremos un nombre para

nuestros usuarios virtuales, dejamos marcada la primera opción y pulsamos **OK**, por defecto la población que se crea es de 10 usuarios aunque podemos incrementar esta cantidad.



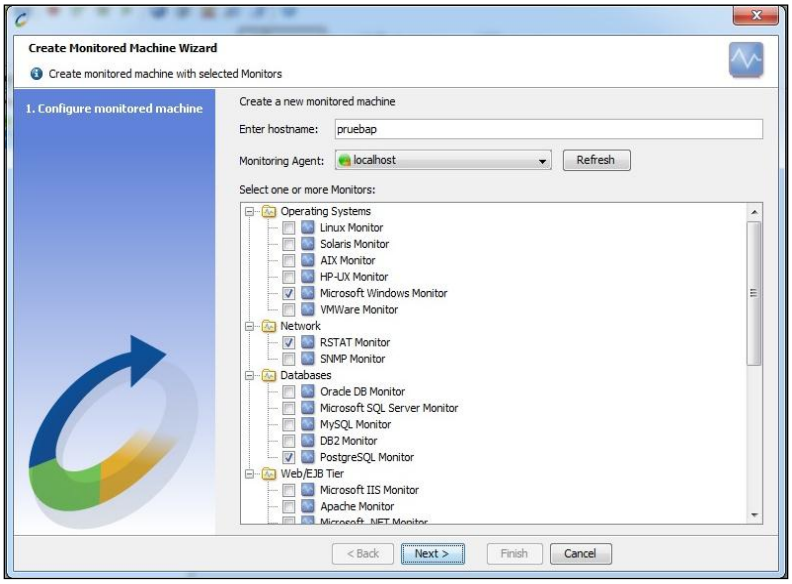
Ingreso de población

Ahora vamos a configurar los monitores de medida para el desempeño, estos monitores nos permiten saber el desempeño de la aplicación en lo que respecta a uso de CPU, memoria, acceso a bases de datos, ancho de banda utilizado y otras opciones, para desplegarlo nos vamos a la pestaña llamada **Monitors** y pulsamos la opción **New Monitor Machine.**



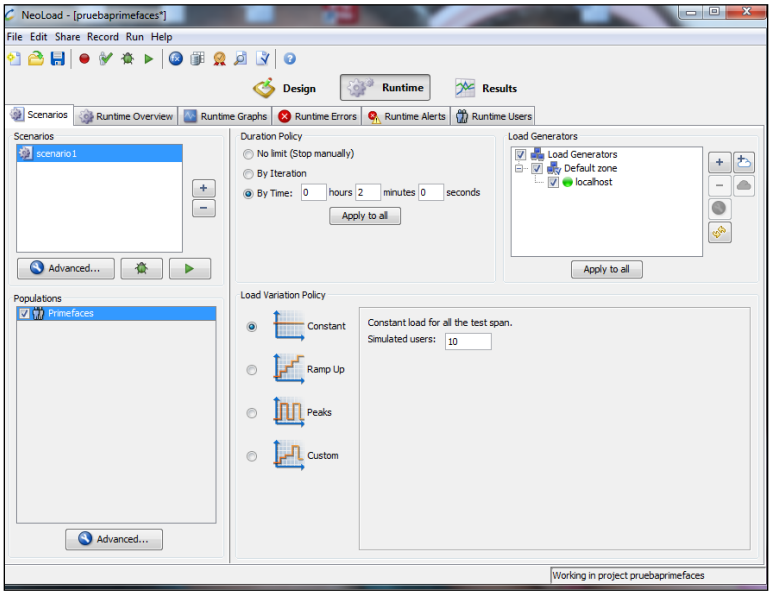
Ingreso de un nuevo Monitor

Aparece una ventana en la cual podemos indicar los monitores que vamos a utilizar, podemos escoger entre monitores para el Sistema Operativo, Red y Bases de Datos, seleccionamos los que necesitamos y pulsamos **Next**.



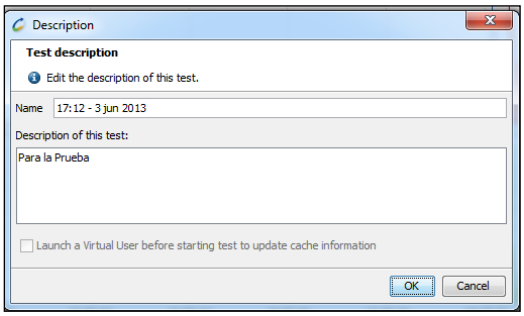
Monitores Escogidos

Para la configuración de ejecución de pruebas, una vez terminado el diseño de la Prueba, nos moveremos al botón **Runtime**, al presionarlo se observará el escenario, la población de usuarios que por defecto será de 10 para la ejecución de las Pruebas, para ello pulsamos el botón **Runtime** ubicado en la parte izquierda de la ventana



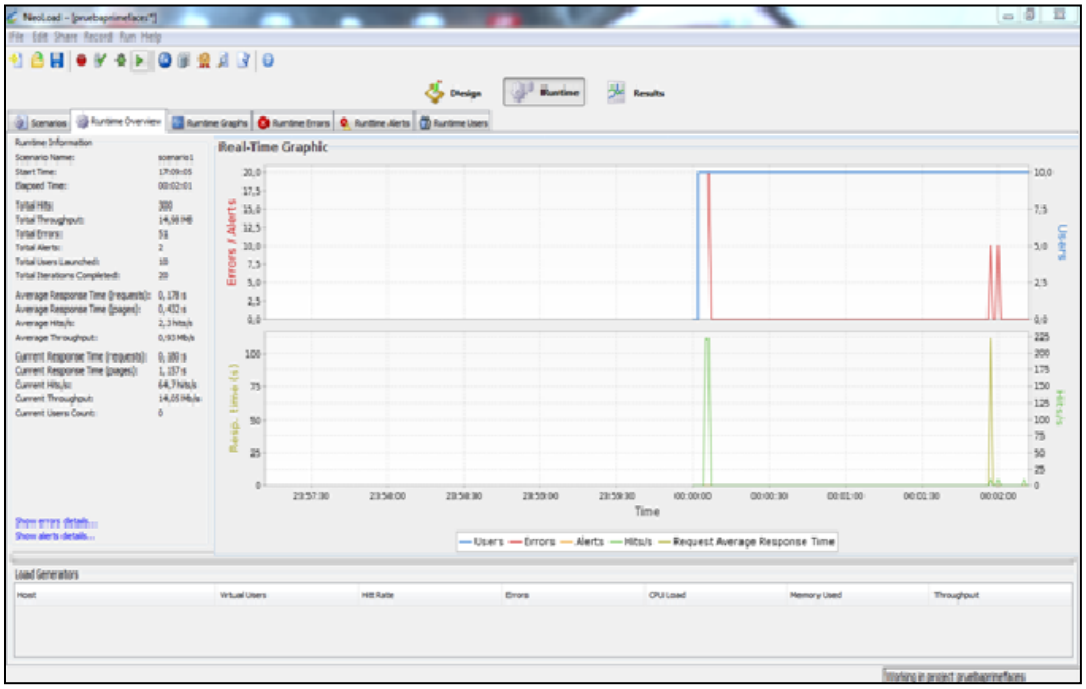
Runtime de las pruebas

Al pulsar **Run** se muestra una ventana emergente que nos pide ingresar un nombre del Reporte de Resultados que por defecto será la fecha y una descripción de la misma, una vez ingresado estos datos pulsamos **OK**.



Ingreso de Nombre y Descripción de la Prueba

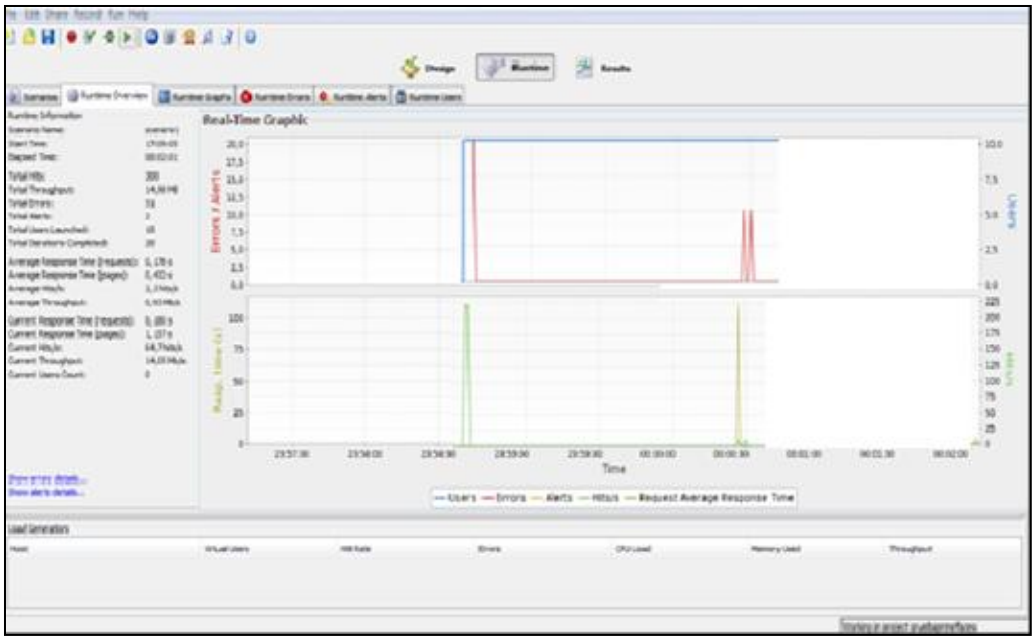
A continuación podremos ver como esta herramienta muestra resultados en tiempo real.



Ejecución de Pruebas Prototipo Primefaces

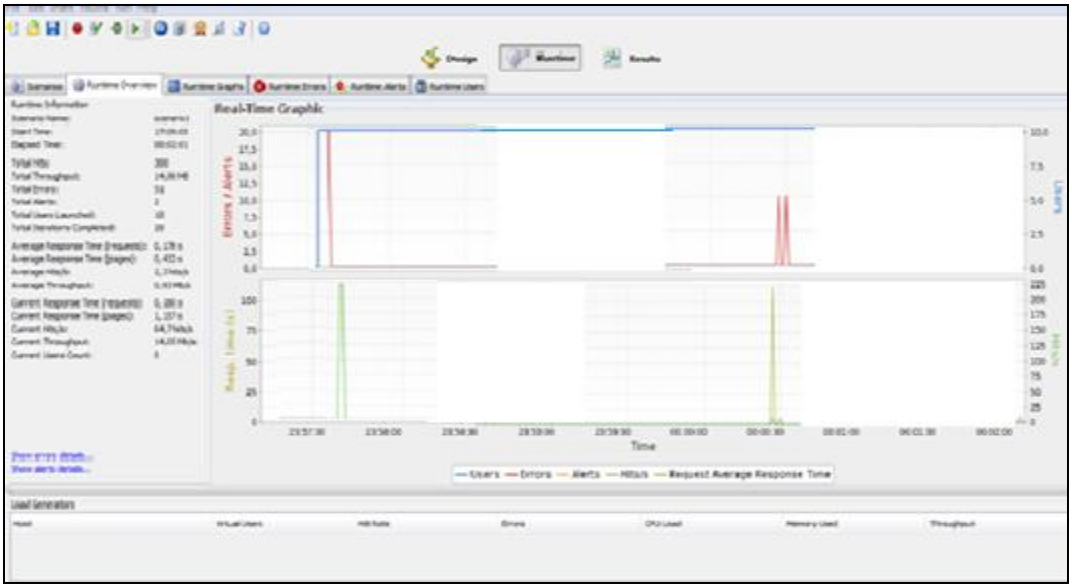
Una vez terminada la Prueba, se puede observar se midió un sin número de parámetros de los cuales se escogerá para el prototipo de Primefaces el tamaño de pagina es HTML:4KB

JS:247 KB CSS:38KB y el tamaño de respuesta Ajax HTML: 2KB JS: 0KB CSS: 0KB que son los parámetros que necesitamos para nuestro estudio .



Ejecución de Pruebas Prototipo Richfaces

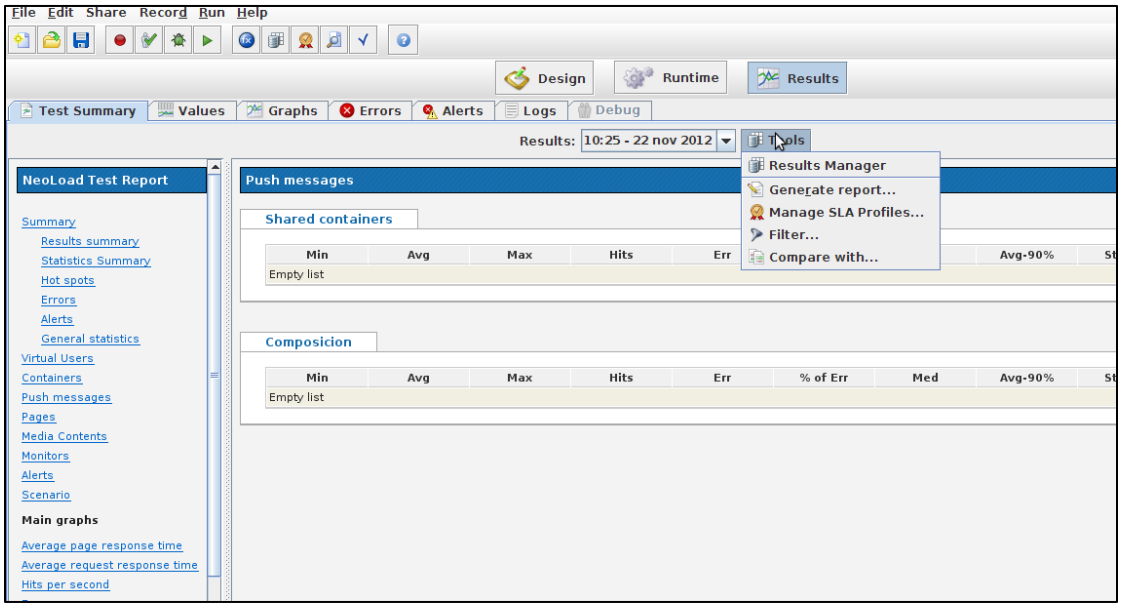
Para el prototipo de Richfaces el tamaño de pagina es HTML: 8KB JS: 275 KB CSS:7KB y el tamaño de respuesta Ajax HTML: 6KB JS: 0KB CSS: 0KB que son los parámetros que necesitamos para nuestro estudio .



Ejecución de Pruebas Prototipo Icefaces

Para el prototipo de Icefaces el tamaño de pagina es HTML: 22KB JS:1275 KB CSS:105KB y el tamaño de respuesta Ajax HTML: 12KB JS: 0KB CSS: 0KB que son los parámetros que necesitamos para nuestro estudio .

Para finalizar al pulsar en el Botón **Results**, la herramienta genera de manera automática un informe detallado acerca de los resultados obtenidos durante la simulación realizada por cada prototipo.



Informe de Resultados de la Prueba.

Anexo 3. Código fuente clases, funciones controladores y paginas xhtml.

Clase

```
package orden.logica.clases;

import master.logica.clases.Dependencia;

import master.logica.clases.Persona;

import master.logica.clases.Usuario;


public class Solicitud {

    private long codigo;

    private Dependencia codigo_dependencia;

    private long codigo_jefe;

    private long codigo_responsable;

    private String nombre_dependencia;

    private String nombre_jefe;

    private String apellido_jefe;

    private String nombre_responsable;

    private String apellido_responsable;

    private long fecha_registro;

    private String motivo;

    private int estado;

    private Usuario codigo_usuario;


    public String getApellido_jefe() {

        return apellido_jefe;    }
```

```
public void setApellido_jefe(String apellido_jefe) {
    this.apellido_jefe = apellido_jefe;    }
public String getApellido_responsable() {
    return apellido_responsable;    }

public void setApellido_responsable(String apellido_responsable) {
    this.apellido_responsable = apellido_responsable;    }

public long getCodigo() {
    return codigo;    }

public void setCodigo(long codigo) {
    this.codigo = codigo;    }

public Dependencia getCodigo_dependencia() {
    return codigo_dependencia;    }

public void setCodigo_dependencia(Dependencia codigo_dependencia) {
    this.codigo_dependencia = codigo_dependencia;    }

public long getCodigo_jefe() {
    return codigo_jefe;    }

public void setCodigo_jefe(long codigo_jefe) {
    this.codigo_jefe = codigo_jefe;    }
```

```
public long getCodigo_responsable() {
    return codigo_responsable;    }

public void setCodigo_responsable(long codigo_responsable) {
    this.codigo_responsable = codigo_responsable;    }

public void setCodigo_responsable(int codigo_responsable) {
    this.codigo_responsable = codigo_responsable;    }

public int getEstado() {
    return estado;    }

public void setEstado(int estado) {
    this.estado = estado;    }

public long getFecha_registro() {
    return fecha_registro;    }

public void setFecha_registro(long fecha_registro) {
    this.fecha_registro = fecha_registro;    }

public String getMotivo() {
    return motivo;    }

public void setMotivo(String motivo) {
```

```
    this.motivo = motivo;    }

public String getNombre_dependencia() {
    return nombre_dependencia;    }

public void setNombre_dependencia(String nombre_dependencia) {
    this.nombre_dependencia = nombre_dependencia;    }

public String getNombre_jefe() {
    return nombre_jefe;    }

public void setNombre_jefe(String nombre_jefe) {
    this.nombre_jefe = nombre_jefe;    }

public String getNombre_responsable() {
    return nombre_responsable;    }

public void setNombre_responsable(String nombre_responsable) {
    this.nombre_responsable = nombre_responsable;    }

public Usuario getCodigo_usuario() {
    return codigo_usuario;    }

public void setCodigo_usuario(Usuario codigo_usuario) {
    this.codigo_usuario = codigo_usuario;    }
```

```
public Solicitud() {
    }

public Solicitud(long codigo, Dependencia codigo_dependencia, long codigo_jefe, long codigo_responsable, String nombre_dependencia, String nombre_jefe, String apellido_jefe, String nombre_responsable, String apellido_responsable, long fecha_registro, String motivo, int estado, Usuario codigo_usuario) {
        this.codigo = codigo;
        this.codigo_dependencia = codigo_dependencia;
        this.codigo_jefe = codigo_jefe;
        this.codigo_responsable = codigo_responsable;
        this.nombre_dependencia = nombre_dependencia;
        this.nombre_jefe = nombre_jefe;
        this.apellido_jefe = apellido_jefe;
        this.nombre_responsable = nombre_responsable;
        this.apellido_responsable = apellido_responsable;
        this.fecha_registro = fecha_registro;
        this.motivo = motivo;
        this.estado = estado;
        this.codigo_usuario = codigo_usuario;
    }
}
```

Funciones

```
package orden.logica.funciones;
```

```
import accesodatos.AccesoDatos;
import accesodatos.ConjuntoResultado;
import accesodatos.Parametro;
import java.sql.SQLException;
import java.util.ArrayList;
import master.logica.clases.Funcion;
import master.logica.funciones.*;
import orden.logica.clases.Solicitud;
import vehiculo_chofer.logica.clases.Licencia;

public class FSolicitud {
public static ArrayList<Solicitud> llenarSolicitud(ConjuntoResultado rs) throws Exception {
 ArrayList<Solicitud> lst = new ArrayList<Solicitud>();
  Solicitud solicitud = null;
  try {
          while (rs.next()) {solicitud = new Solicitud(rs.getLong("pcodigo"),
          FDependencia.ObtenerDependenciaDadocodigo(rs.getInt("pcodigo_dependencia")),
                  rs.getLong("pcodigo_jefe"), rs.getLong("pcodigo_responsable"),
                  rs.getString("pnombre_dependencia"), rs.getString("pnombre_jefe"),
                  rs.getString("papellido_jefe"), rs.getString("pnombre_responsable"),
                  rs.getString("papellido_responsable"), rs.getLong("pfecha_registro"),
                  rs.getString("pmotivo"),
                 rs.getInt("pestado"),FUsuario.ObtenerUsuarioDadoCodigo(rs.getLong("pcodigo_usu
                 ario")));
            lst.add(solicitud);          }
```

```
        } catch (Exception e) {
            lst.clear();
            throw e;
        }
    return lst;    }

public static ArrayList<Solicitud> ObtenerSolicitud() throws Exception {
        ArrayList<Solicitud> lst = new ArrayList<Solicitud>();
        try {
            String sql = "select * from orden.f_select_solicitud()";
            ConjuntoResultado rs = AccesoDatos.ejecutaQuery(sql);
            lst = llenarSolicitud(rs);
            rs = null;
        } catch (SQLException exConec) {
            throw new Exception(exConec.getMessage());}
        return lst;    }

public static ArrayList<Solicitud> ObtenerSolicitudesActivasSinAsignacion() throws
Exception {
        ArrayList<Solicitud> lst = new ArrayList<Solicitud>();
        try {
            String sql = "select * from orden.f_select_solicitudes_activas_sin_asignacion()";
            ConjuntoResultado rs = AccesoDatos.ejecutaQuery(sql);
            lst = llenarSolicitud(rs);
            rs = null;
```

```
        } catch (SQLException exConec) {
            throw new Exception(exConec.getMessage());
        }
        return lst;
    }

public static ArrayList<Solicitud> ObtenerSolicitudesActivas() throws Exception {
        ArrayList<Solicitud> lst = new ArrayList<Solicitud>();
        try {
            String sql = "select * from orden.f_select_solicitudes_activas()";
            ConjuntoResultado rs = AccesoDatos.ejecutaQuery(sql);
            lst = llenarSolicitud(rs);
            rs = null;
        } catch (SQLException exConec) {
            throw new Exception(exConec.getMessage());
        }
        return lst;
    }

public static ArrayList<Solicitud> ObtenerSolicitudEstadoPendiente() throws Exception {
        ArrayList<Solicitud> lst = new ArrayList<Solicitud>();
        try {
            String sql = "select * from orden.f_select_solicitud_estado_pendiente()";
            ConjuntoResultado rs = AccesoDatos.ejecutaQuery(sql);
            lst = llenarSolicitud(rs);
```

```
            rs = null;
        } catch (SQLException exConec) {
            throw new Exception(exConec.getMessage());
        }
        return lst;    }

public static Solicitud ObtenerSolicitudDadoCodigo(long codigo) throws Exception {
        Solicitud lst;
        try {
            ArrayList<Parametro> lstP = new ArrayList<Parametro>();
            String sql = "select * from orden.f_select_solicitud_dado_codigo(?)";
            lstP.add(new Parametro(1, codigo));
            ConjuntoResultado rs = AccesoDatos.ejecutaQuery(sql, lstP);
            lst = new Solicitud();
            lst = llenarSolicitud(rs).get(0);
            rs = null;
        } catch (SQLException exConec) {
            throw new Exception(exConec.getMessage());
        }
        return lst;
    }
public static boolean cancelarSolicitudesDadoCodigo(long codigo) throws Exception {
        boolean eje = false;
        try {
            ArrayList<Parametro> lstP = new ArrayList<Parametro>();
```

```
        String sql = "select * from orden.f_update_solicitud_estado_cancelado(?)";
        lstP.add(new Parametro(1, codigo));
        ConjuntoResultado rs = AccesoDatos.ejecutaQuery(sql, lstP);
        while (rs.next()) {
            if (rs.getString(0).equals("true"));
            eje = true;
        }
    } catch (SQLException exConec) {
        throw new Exception(exConec.getMessage());
    }
    return eje;
}

public static boolean insertar(Solicitud solicitud) throws Exception {
    boolean eje = false;
    try {
        ArrayList<Parametro> lstP = new ArrayList<Parametro>();
        String sql = "select * from orden.f_insert_solicitud(?,?,?,?,?,?,?,?,?,?,?,?,?)";
        lstP.add(new Parametro(1, solicitud.getCodigo_dependencia().getCodigo()));
        lstP.add(new Parametro(2, solicitud.getCodigo_jefe()));
        lstP.add(new Parametro(3, solicitud.getCodigo_responsable()));
        lstP.add(new Parametro(4, solicitud.getCodigo_dependencia().getNombre()));
        lstP.add(new Parametro(5, solicitud.getNombre_jefe()));
        lstP.add(new Parametro(6, solicitud.getApellido_jefe()));
        lstP.add(new Parametro(7, solicitud.getNombre_responsable()));
```

```
            lstP.add(new Parametro(8, solicitud.getApellido_responsable()));
            lstP.add(new Parametro(9, solicitud.getFecha_registro()));
            lstP.add(new Parametro(10, solicitud.getMotivo()));
            lstP.add(new Parametro(11, solicitud.getEstado()));
            lstP.add(new Parametro(12, solicitud.getCodigo_usuario().getCodigo()));
            ConjuntoResultado rs = AccesoDatos.ejecutaQuery(sql, lstP);
            while (rs.next()) {
                if (rs.getString(0).equals("true"));
                eje = true;
            }
        } catch (SQLException exConec) {
            throw new Exception(exConec.getMessage());
        }
        return eje;
    }

public static boolean actualizar(Solicitud solicitud) throws Exception {
        boolean eje = false;
        try {
            ArrayList<Parametro> lstP = new ArrayList<Parametro>();
            String sql = "select * from orden.f_update_solicitud(?,?,?,?,?,?,?,?,?,?,?,?,?)";
            lstP.add(new Parametro(1, solicitud.getCodigo_dependencia().getCodigo()));
            lstP.add(new Parametro(2, solicitud.getCodigo_jefe()));
            lstP.add(new Parametro(3, solicitud.getCodigo_responsable()));
            lstP.add(new Parametro(4, solicitud.getCodigo_dependencia().getNombre()));
```

```
        lstP.add(new Parametro(5, solicitud.getNombre_jefe()));
        lstP.add(new Parametro(6, solicitud.getApellido_jefe()));
        lstP.add(new Parametro(7, solicitud.getNombre_responsable()));
        lstP.add(new Parametro(8, solicitud.getApellido_responsable()));
        lstP.add(new Parametro(9, solicitud.getFecha_registro()));
        lstP.add(new Parametro(10, solicitud.getMotivo()));
        lstP.add(new Parametro(11, solicitud.getEstado()));
        lstP.add(new Parametro(12, solicitud.getCodigo_usuario().getCodigo()));
        lstP.add(new Parametro(13, solicitud.getCodigo()));
        ConjuntoResultado rs = AccesoDatos.ejecutaQuery(sql, lstP);
        while (rs.next()) {
            if (rs.getString(0).equals("true"));
            eje = true;
        }
    } catch (SQLException exConec) {
        throw new Exception(exConec.getMessage());
    }
    return eje;    }}
```

Controlador

```
package orden.presentacion.beans;

import com.lowagie.text.*;
import java.io.File;
```

```
import java.io.IOException;
import java.sql.Timestamp;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.annotation.PostConstruct;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ManagedProperty;
import javax.faces.bean.ViewScoped;
import javax.faces.context.FacesContext;
import javax.servlet.ServletContext;
import master.logica.clases.*;
import master.logica.funciones.*;
import master.presentacion.beans.SesionUsuarioDataManager;
import orden.logica.clases.*;
import orden.logica.funciones.*;
import org.apache.poi.hssf.usermodel.*;
import org.apache.poi.hssf.util.HSSFColor;
import org.primefaces.context.DefaultRequestC
import org.primefaces.model.LazyDataModel;
import recursos.Util;
```

```
@ManagedBean
@ViewScoped
public class solicitudVehiculosC {

@ManagedProperty(value = "#{sesionUsuarioDataManager}")
private SesionUsuarioDataManager sesionUsuarioDM;
private Solicitud selecionarSolicitud;
private Solicitud selecionarPendientes;
private ArrayList<Solicitud> solicitudes;
private ArrayList<Solicitud> solicitudesPendientes;
private Solicitud solicitud;
private Date date1;
private int tipoDependencia;
private ArrayList<Dependencia> dependencias;
private String criterioBusquedaPersona;
private String datoBusqueda;
private ArrayList<Persona> personas;
private LazyDataModel<Persona> lazyModel;
private Persona selectedPersona;
private Persona persona;
private PersonaDependenciaCargoDecano personasDep;
private int tipoEstado;

public SesionUsuarioDataManager getSesionUsuarioDM() {
        return sesionUsuarioDM;    }
```

```
public void setSesionUsuarioDM(SesionUsuarioDataManager sesionUsuarioDM) {
        this.sesionUsuarioDM = sesionUsuarioDM;     }

public Solicitud getSelecionarSolicitud() {
        return selecionarSolicitud;     }

public void setSelecionarSolicitud(Solicitud selecionarSolicitud) {
        this.selecionarSolicitud = selecionarSolicitud;     }

public ArrayList<Solicitud> getSolicitudes() {
        return solicitudes;    }

public void setSolicitudes(ArrayList<Solicitud> solicitudes) {
        this.solicitudes = solicitudes;     }

public Solicitud getSolicitud() {
        return solicitud;     }

public void setSolicitud(Solicitud solicitud) {
        this.solicitud = solicitud;     }

public Date getDate1() {
        return date1;     }

public void setDate1(Date date1) {
```

```
        this.date1 = date1;    }

public int getTipoDependencia() {
        return tipoDependencia;    }

public void setTipoDependencia(int tipoDependencia) {
        this.tipoDependencia = tipoDependencia;    }

public ArrayList<Dependencia> getDependencias() {
        return dependencias;    }

public void setDependencias(ArrayList<Dependencia> dependencias) {
        this.dependencias = dependencias;    }

public String getDatoBusqueda() {
        return datoBusqueda;    }

public void setDatoBusqueda(String datoBusqueda) {
        this.datoBusqueda = datoBusqueda;    }

public String getCriterioBusquedaPersona() {
        return criterioBusquedaPersona;    }

public void setCriterioBusquedaPersona(String criterioBusquedaPersona) {
        this.criterioBusquedaPersona = criterioBusquedaPersona;    }
```

```
public LazyDataModel<Persona> getLazyModel() {
        return lazyModel;    }

public void setLazyModel(LazyDataModel<Persona> lazyModel) {
        this.lazyModel = lazyModel;    }

public Persona getPersona() {
        return persona;    }

public void setPersona(Persona persona) {
        this.persona = persona;    }

public ArrayList<Persona> getPersonas() {
        return personas;    }

public void setPersonas(ArrayList<Persona> personas) {
        this.personas = personas;    }

public Persona getSelectedPersona() {
        return selectedPersona;    }

public void setSelectedPersona(Persona selectedPersona) {
        this.selectedPersona = selectedPersona;    }

public PersonaDependenciaCargoDecano getPersonasDep() {
```

```
        return personasDep;     }

public void setPersonasDep(PersonaDependenciaCargoDecano personasDep) {
        this.personasDep = personasDep;     }

public int getTipoEstado() {
        return tipoEstado;     }
public void setTipoEstado(int tipoEstado) {
        this.tipoEstado = tipoEstado;     }

public Solicitud getSelecionarPendientes() {
        return selecionarPendientes;     }

public void setSelecionarPendientes(Solicitud selecionarPendientes) {
        this.selecionarPendientes = selecionarPendientes;     }

public ArrayList<Solicitud> getSolicitudesPendientes() {
        return solicitudesPendientes;     }

public void setSolicitudesPendientes(ArrayList<Solicitud> solicitudesPendientes) {
        this.solicitudesPendientes = solicitudesPendientes;     }

public solicitudVehiculosC() {
        selecionarSolicitud = new Solicitud();
        solicitudes = new ArrayList<Solicitud>();
```

```
        dependencias = new ArrayList<Dependencia>();
        solicitud = new Solicitud();
        date1 = new Date();
        tipoEstado = 1;
    }

@PostConstruct
public void postSolicitudVehiculosC() {
        cargarSolicitudes();
        cargarDependencias();
        cargarSolicitudesPendientes();
        selecionarSolicitud = solicitudes.get(0);
        selecionarPendientes=solicitudesPendientes.get(0);
    }

public void reinit()    {
        solicitud = new Solicitud();
        selecionarSolicitud = new Solicitud();
        solicitudes = new ArrayList<Solicitud>();
        cargarSolicitudes();
        cargarDependencias();
        cargarSolicitudesPendientes();
        selecionarSolicitud = solicitudes.get(0);
        selecionarPendientes=solicitudesPendientes.get(0);
    }
```

```
public void cargarSolicitudes() {
        try {
            this.solicitudes = FSolicitud.ObtenerSolicitud();
            System.out.printf("Datos Cargados de las Solicitudes.");
        } catch (Exception e) {
           Util.addErrorMessage("Error al cargar datos de las Solicitudes" );
        }    }


public void cargarSolicitudesPendientes() {
        try {
            this.solicitudesPendientes = FSolicitud.ObtenerSolicitudEstadoPendiente();
            System.out.printf("Datos Cargados de las Solicitudes .");
        } catch (Exception e) {
           Util.addErrorMessage("Error al cargar datos de las Solicitudes " );
        }
    }
public void cargarDependencias() {
        try {
            this.dependencias = FDependencia.ObtenerDependencia();
            System.out.printf("Datos Cargados de las Dependencias.");
        } catch (Exception e) {
            Util.addErrorMessage("Error al cargar datos de las Dependencias " );
        }
    }
```

```
public void cargarPersonasDependencias() {
        try {
           this.personasDep =
          FPersonaDependenciaCargoDecano.ObtenerPersonaDadoDependencia(tipoDependencia);
           System.out.printf("Datos Cargados de los Jefes de Dependencia.");
        } catch (Exception e) {
           Util.addErrorMessage("Error al cargar datos de los Jefes de Dependencia "); }
    }

public String obtieneFecha(long fecha) {
        Timestamp fecha1 = null;
        try {
           fecha1 = recursos.Tools.obtieneFechaTimeStamp(fecha);
        } catch (ParseException e) {
           Logger.getLogger(solicitudVehiculosC.class.getName()).log(Level.SEVERE, null, e);
        }
        SimpleDateFormat formato = new SimpleDateFormat("dd-MM-yyyy");
        return formato.format(fecha1.getTime());    }

public void insertarSolicitud1() {
        try {
           if (tipoDependencia > 0 && !(solicitud.getMotivo().equals("") && tipoEstado >= 0)) {
solicitud.setCodigo_dependencia(FDependencia.ObtenerDependenciaDadocodigo(tipoDependencia
));          solicitud.setCodigo_jefe(personasDep.getCodigo_persona());
              solicitud.setApellido_jefe(personasDep.getApellido());
```

```
            solicitud.setNombre_jefe(personasDep.getNombre());
            solicitud.setCodigo_responsable(persona.getCodigo_persona());
            solicitud.setApellido_responsable(persona.getApellido());
            solicitud.setNombre_responsable(persona.getNombre());
            solicitud.setFecha_registro(date1.getTime() / 1000);
            solicitud.setEstado(2);
           solicitud.setCodigo_usuario(FUsuario.ObtenerUsuarioDadoCodigo(sesionUsuarioDM.get
           SesionUsuario().getCodigo()));
           if (FSolicitud.insertar(solicitud)) {
               Util.addSuccessMessage("DATOS INSERTADOS");
               reinit();
               DefaultRequestContext.getCurrentInstance().execute("wdlgNuevaSolicitud.hide();");
            } else {
               Util.addErrorMessage("ERROR EN LA INSERCION");
            }
         } else {
            Util.addErrorMessage("COMPLETE LA INFORMACIÓN REQUERIDA");
         }
      } catch (Exception e) {
         Util.addErrorMessage("ERROR EN LA INSERCION");       }    }
```

public void insertarSolicitud() {

try {

selecionarPendientes.setCodigo_usuario(FUsuario.ObtenerUsuarioDadoCodigo(sesionUsuarioDM.

getSesionUsuario().getCodigo()));

```
    selecionarPendientes.setEstado(1);
        if (FSolicitud.actualizar(selecionarPendientes)) {
            Util.addSuccessMessage("SOLICITUD APROBADA");
            DefaultRequestContext.getCurrentInstance().execute("wdlgActualizarSolicitud.hide();");
           cargarSolicitudesPendientes();
        } else {
            Util.addErrorMessage("ERROR AL ALMACENAR DATOS");
        }
      } catch (Exception e) {
      Util.addErrorMessage("ERROR EN LA ACTUALIZACION");
    }
  }

public void obtenerPersonas() {
      try {
         if (criterioBusquedaPersona.equals("cedula")) {
            setPersonas(FPersona.ObtenerPersonaDadoCedula(datoBusqueda));
         } else {
            setPersonas(FPersona.ObtenerPersonaDadoApellido(datoBusqueda.toUpperCase()));
         }
         setLazyModel(new FLazyPersonasDataModel(getPersonas()));
      } catch (Exception e) {
       Util.addErrorMessage("ERROR EN LA OBTENCION DE PERSONAS");
              }  }
```

```
public void asignarSelectedPersonaR() {
        try {
            persona = (FPersona.ObtenerPersonaDadoCodigo(selectedPersona.getCodigo_persona()));
            System.out.print(getPersona().getCedula());
        } catch (java.lang.Exception e) {
            System.out.print(getPersona().getApellido() + " " + e.getMessage());
        }
        DefaultRequestContext.getCurrentInstance().execute("wdlBusquedaPersonas.hide()");
        DefaultRequestContext.getCurrentInstance().execute("wdlgNuevaSolicitud.show()");
    }

public void postProcessXLS(Object document) {
        HSSFWorkbook wb = (HSSFWorkbook) document;
        HSSFSheet sheet = wb.getSheetAt(0);
        HSSFRow header = sheet.getRow(0);

        HSSFCellStyle cellStyle = wb.createCellStyle();
        cellStyle.setFillForegroundColor(HSSFColor.GREY_40_PERCENT.index);
        cellStyle.setFillPattern(HSSFCellStyle.SOLID_FOREGROUND);

        for (int i = 0; i < header.getPhysicalNumberOfCells(); i++) {
            ((HSSFCell) header.getCell(i)).setCellStyle(cellStyle);
        }
    }
```

```
public void preProcessPDF(Object document) throws IOException, BadElementException,
DocumentException {
        Document pdf = (Document) document;
        pdf.setPageSize(PageSize.A4.rotate());
        pdf.open();
        ServletContext servletContext = (ServletContext)
        FacesContext.getCurrentInstance().getExternalContext().getContext();
        String logo = servletContext.getRealPath("") + File.separator + "resources" + File.separator +
        "logo" + File.separator + "logo4.png";
        pdf.add(Image.getInstance(logo));
    }
}
```

Página Xhtml

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.org/ui"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:f="http://java.sun.com/jsf/core">
    <h:head>
        <title>
```

```
        <ui:define name="tituloNavegador">Gestión de una Solicitud</ui:define>
    </title>
</h:head>
<h:body>
    <ui:composition template="../../templates/plantillaInternaCenter.xhtml">
        <ui:define name="top">Gestión de una Solicitud</ui:define>
        <ui:define name="contenido">
            <p:growl id="mensajes" autoUpdate="true"/>
            <h:form id="frmSolicitud">
                <p:ajaxStatus onstart="dlgStatus.show();" oncomplete="dlgStatus.hide();"/>
                <p:panel id="pnlSolicitud" >
                     <p:dataTable id="tblSolicitud" value="#{solicitudVehiculosC.solicitudes}"
                    var="solicitud"  paginator="true" rows="10" draggableColumns="false"
                    rowKey="#{solicitud.codigo}" emptyMessage="NO SE ECONTRARON
                    DATOS"  selectionMode="single" selection="#{solicitudVehiculosC.
                    selecionarSolicitud}" >
                    <p:column filterBy="#{solicitud.nombre_dependencia}">
                        <f:facet name="header">
                            <h:outputText style="width:140px" value="NOMBRE DEPENDENCIA"/>
                        </f:facet>
                             <h:outputText style="width:140px"
                            value="#{solicitud.nombre_dependencia}"/>
                    </p:column>
                    <p:column >
                        <f:facet name="header">
```

```
        <h:outputText style="width:140px" value="APELLIDOS Y NOMBRES
       DEL JEFE"/>
    </f:facet>
        <h:outputText style="width:140px" value="#{solicitud.apellido_jefe}
       #{solicitud.nombre_jefe}"/>
</p:column>
<p:column >
    <f:facet name="header">
       <h:outputText style="width:140px" value="APELLIDOS Y NOMBRES
       DEL RESPONSABLE"/>
    </f:facet>
      <h:outputText  style="width:140px" value="#{solicitud.apellido_responsable}
     #{solicitud.nombre_responsable}"/>
</p:column>
<p:column >
    <f:facet name="header">
       <h:outputText style="width:140px" value="FECHA REGISTRO"/>
    </f:facet>
     <h:outputText style="width:140px"
     value="#{solicitudVehiculosC.obtieneFecha(solicitud.fecha_registro)} "/>
</p:column>
<p:column>
    <f:facet name="header">
       <h:outputText style="width:140px" value="MOTIVO"/>
    </f:facet>
```

```
        <h:outputText style="width:140px" value="#{solicitud.motivo}"/>
    </p:column>
    <p:column >
        <f:facet name="header">
            <h:outputText style="width:140px" value="ESTADO"/>
        </f:facet>
        <h:outputText style="width:140px"
      value="#{solicitud.estado==0?'RECHAZADO'
      :solicitud.estado==1?'APROBADO':solicitud.estado==2?'PENDIENTE':solicitu
      d.estado==3?'CANCELADO':solicitud.estado==4?'ASIGNADO':'FINALIZAD
      O'}"/>
    </p:column>
    <f:facet name="footer" >
        <p:commandButton id="btnNuevaSolicitud" value="Nuevo" icon="ui-icon-
         newwin"  onclick="wdlgNuevaSolicitud.show()"
         update=":frmNuevaSolicitud:pngDependencia
                  :frmNuevaSolicitud:pngSolicitudResponsable
                  :frmNuevaSolicitud:pnSolicitud"/>
        <p:commandButton id="btnVerSolicitud" value="Ver" icon="ui-icon-newwin"
                  onclick="wdlgVerSolicitud.show()"
                  />
    </f:facet>
</p:dataTable>
<h:panelGrid columns="2">
    <p:panel header="TODOS LOS DATOS">
```

```
    <h:commandLink>
        <p:graphicImage value="/resources/images/excel_1.png" />
        <p:dataExporter type="xls" target="tblSolicitud"
                        fileName="Solicitud" encoding="ISO-8859-1"
                        postProcessor="#{solicitudVehiculosC.postProcessXLS}"/>
    </h:commandLink>

    <h:commandLink>
        <p:graphicImage value="/resources/images/pdf.png" />
        <p:dataExporter type="pdf" target="tblSolicitud"
                        fileName="Solicitud" encoding="ISO-8859-1"
                        preProcessor="#{solicitudVehiculosC.preProcessPDF}"/>
    </h:commandLink>
</p:panel>
<p:panel header="DATOS DE ESTA PAGINA">
    <h:commandLink>
        <p:graphicImage value="/resources/images/excel_1.png" />
        <p:dataExporter type="xls" target="tblSolicitud"
                        fileName="Solicitud" encoding="ISO-8859-1"
                        pageOnly="true"
        postProcessor="#{solicitudVehiculosC.postProcessXLS}"/>
    </h:commandLink>
    <h:commandLink>
        <p:graphicImage value="/resources/images/pdf.png" />
        <p:dataExporter type="pdf" target="tblSolicitud"
```

```
                                    fileName="Solicitud"
                                    pageOnly="true" encoding="ISO-8859-1"
                                    preProcessor="#{solicitudVehiculosC.preProcessPDF}"/>
                        </h:commandLink>
                    </p:panel>
                </h:panelGrid>
            </p:panel>
        </h:form>
    </ui:define>
  <ui:define name="dialogos"
      <p:dialog modal="true" widgetVar="dlgStatus" header="Procesando" draggable="false"
          closable="false"  resizable="false">
              <p:graphicImage value="/resources/images/ajaxloadingbar.gif" />
          </p:dialog>
<p:dialog id="dlgNuevaSolicitud" widgetVar="wdlgNuevaSolicitud" header="Nueva Solicitud"
    modal="true" closable="false" resizable="false" draggable="true"
showEffect="clip" hideEffect="fold">
  <h:form id="frmNuevaSolicitud">
     <p:ajaxStatus onstart="dlgStatus.show();" oncomplete="dlgStatus.hide();"/>
       <p:messages id="messages" autoUpdate="true"/>
          <p:panel header="Dependencia">
            <h:panelGrid id="pngDependencia" columns="2" cellpadding="5">
               <h:outputText value="Dependencia"/>
                <p:selectOneMenu value="#{solicitudVehiculosC.tipoDependencia}">
```

```
<f:selectItem itemLabel="--Seleccione Dependencia--"
    itemValue="#{-1}"/>
 <f:selectItems value="#{solicitudVehiculosC.dependencias}"
 var="depe"  itemLabel="#{depe.nombre}"
 itemValue="#{depe.codigo}"/
  <p:ajax event="change"
 listener="#{solicitudVehiculosC.cargarPersonasDependencias()}" update="nombre
 apellido" />
 </p:selectOneMenu>
<h:outputText value="Nombre"/>
<p:inputText id="nombre" value="#{solicitudVehiculosC.personasDep.nombre}"
    disabled="true"/>
<h:outputText value="Apellido"/>
<p:inputText  id="apellido" value="#{solicitudVehiculosC.personasDep.apellido}"
disabled="true"/>
</h:panelGrid>
  </p:panel>
  <p:panel header="Responsable">
    <h:panelGrid id="pngSolicitudResponsable" columns="2" cellpadding="5">
      <h:outputText value="Busque un Responsable:"/>
      <p:commandButton value="Buscar" onclick="wdlBusquedaPersonas.show()"
       process="@this"/>
      <h:outputText value="Nombre"/>
      <p:inputText id="txtNombre" value="#{solicitudVehiculosC.persona.nombre}"
       disabled="true"/>
```

```
        <h:outputText value="Apellido"/>
                                    <p:inputText id="txtApellido"
         value="#{solicitudVehiculosC.persona.apellido}" disabled="true"/>
    </h:panelGrid>
</p:panel>
<p:panel header="Solicitud">
    <h:panelGrid id="pnSolicitud" columns="2" cellpadding="5">
        <h:outputText value="Motivo"/>
        <p:inputText id="motivo" value="#{solicitudVehiculosC.solicitud.motivo}"/>
        <h:outputText value="Fecha de Registro"/>
        <p:calendar id="calen" value="#{solicitudVehiculosC.date1}"/>
        <h:outputText value="Estado" id="estado"/>
        <p:selectOneMenu value="#{solicitudVehiculosC.tipoEstado}" id="estadoS">
            <f:selectItem itemLabel="Rechazado" itemValue="0" />
            <f:selectItem itemLabel="Activo" itemValue="1" />
            <f:selectItem itemLabel="Pendiente" itemValue="2" />
        </p:selectOneMenu>
    </h:panelGrid>
</p:panel>
<p:commandButton id="btnAceptarNuevaS" value="Aceptar"
                action="#{solicitudVehiculosC.insertarSolicitud1()}"
                update=":frmSolicitud:pnlSolicitud"
                process="@form nombre apellido motivo estadoS"/>
<p:commandButton id="btnCancelarNuevaSolicitud" value="Cancelar"
                onclick="wdlgNuevaSolicitud.hide();"/>
```

```
    </h:form>
</p:dialog>
<p:dialog id="dlgVerSolicitud" widgetVar="wdlgVerSolicitud" header="Ver Solicitud"
          modal="true" closable="false" resizable="false" draggable="true"
          showEffect="clip" hideEffect="fold">
    <h:form id="frmVerSolicitud">
        <p:ajaxStatus onstart="dlgStatus.show();" oncomplete="dlgStatus.hide();"/>
        <p:messages id="messages" autoUpdate="true"/>
        <h:panelGrid id="pnVerSolicitud" columns="2" cellpadding="5">
            <h:outputText value="Dependencia"/>
              <p:inputText
              value="#{solicitudVehiculosC.selecionarSolicitud.nombre_dependencia}"
              disabled="true"/>
            <h:outputText value="Jefe"/>
               <p:inputText  value="#{solicitudVehiculosC.selecionarSolicitud.nombre_jefe}
              #{solicitudVehiculosC.selecionarSolicitud.apellido_jefe}" disabled="true"/>
            <h:outputText value="Responsable"/>
              <p:inputText
              value="#{solicitudVehiculosC.selecionarSolicitud.nombre_responsable}
              #{solicitudVehiculosC.selecionarSolicitud.apellido_responsable}"
              disabled="true"/>
            <h:outputText value="Motivo"/>
            <p:inputText  value="#{solicitudVehiculosC.selecionarSolicitud.motivo}"
              disabled="true"/>
            <h:outputText value="Fecha de Registro"/>
```

```
            <p:inputText  value="#{
            solicitudVehiculosC.obtieneFecha(solicitudVehiculosC.selecionarSolicitud.fecha_
            registro)}" disabled="true"/>
            <h:outputText value="Estado" id="estado"/>
             <p:inputText
            value="#{solicitudVehiculosC.selecionarSolicitud.estado==0?'RECHAZADO'
            :solicitudVehiculosC.selecionarSolicitud.estado==1?'APROBADO'
            :solicitudVehiculosC.selecionarSolicitud.estado==2?'PENDIENTE':
            solicitudVehiculosC.selecionarSolicitud.estado==3?'CANCELADO'
            :solicitudVehiculosC.selecionarSolicitud.estado==4?'ASIGNADO':'FINALIZAD
            O'}" disabled="true"/>
        </h:panelGrid>
        <p:commandButton id="btnCancelarVerSolicitud" value="Regresar"
       onclick="wdlgVerSolicitud.hide();"
                        update=":frmSolicitud:pnlSolicitud" type="reset"/>
    </h:form>
</p:dialog>
<p:dialog header="Búsqueda de Personas"  widgetVar="wdlBusquedaPersonas"
          id="dlgBusquedaPersonas" resizable="false" modal="true">
    <p:panel id="pnlContenedorBusquedaPersonas">
        <h:form id="formPersonas">
            <p:ajaxStatus onstart="dlgStatus.show();" oncomplete="dlgStatus.hide();"/>
            <p:panelGrid id="pngBusquedaPersona" columns="4">
                <h:outputText id="txtBuscar" value="Buscar por:"/>
```

```
    <p:selectOneMenu id="cmbCriterioBusqueda"
   value="#{solicitudVehiculosC.criterioBusquedaPersona}"
                style="width: 150px;">
       <f:selectItem itemValue="#{null}" itemLabel="--Elija una opción--"/>
       <f:selectItem itemValue="cedula" itemLabel="CEDULA"/>
       <f:selectItem itemValue="apellido" itemLabel="APELLIDOS"/>
    </p:selectOneMenu>
    <p:inputText id="txtBusca" value="#{solicitudVehiculosC.datoBusqueda}"/>
    <p:commandButton value="Aceptar" update="pTabla"
                action="#{solicitudVehiculosC.obtenerPersonas()}"
                process="@form cmbCriterioBusqueda" />
</p:panelGrid>
<p:panel id="pTabla">
    <p:ajaxStatus onstart="dlgStatus.show();" oncomplete="dlgStatus.hide();"/>
    <p:dataTable var="persona" value="#{solicitudVehiculosC.lazyModel}"
     paginator="true" rows="5"
     paginatorTemplate="{RowsPerPageDropdown} {FirstPageLink}
     {PreviousPageLink}
                {CurrentPageReport} {NextPageLink} {LastPageLink}"
                rowsPerPageTemplate="5,10" selectionMode="single"
                selection="#{solicitudVehiculosC.selectedPersona}"
                id="personaTable" sortBy="#{persona.apellido}"
                rowKey="#{persona.codigo_persona}">
       <p:column headerText="Código" sortBy="#{persona.codigo_persona}">
          <h:outputText value="#{persona.codigo_persona}" />
```

```
                    </p:column>
                    <p:column headerText="Cédula" sortBy="#{persona.codigo_persona}">
                        <h:outputText value="#{persona.cedula}" />
                    </p:column>
                    <p:column headerText="Apellidos" sortBy="#{persona.apellido}">
                        <h:outputText value="#{persona.apellido}" />
                    </p:column>
                    <p:column headerText="Nombre" sortBy="#{persona.nombre}">
                        <h:outputText value="#{persona.nombre}"/>
                    </p:column>
                </p:dataTable>
                 <p:commandButton value="Aceptar"
                 action="#{solicitudVehiculosC.asignarSelectedPersonaR()}"
                  update=":frmNuevaSolicitud:txtNombre :frmNuevaSolicitud:txtApellido"/>
            </p:panel>
        </h:form>
    </p:panel>
</p:dialog>
</ui:define>
</ui:composition>
</h:body>
</html>
```

BIBLIOGRAFÍA

1. JAVASERVER FACES

http://www.desarrolloweb.com/articulos/2380.php

http://www.programacion.com/articulo/introduccion_a_la_tecnologia_javaserver_faces233

[2012/10/15]

http://kalistog.wordpress.com/javaserver-faces-jsf/

http://jansoblog.wordpress.com/2007/10/23/ciclo-de-vida-en-jsf/

[2012/10/20]

2. AJAX

http://www.librosweb.es/ajax/

[2012/11/04]

3. RICHFACES

http://www.jboss.org/richfaces

http://showcase.richfaces.org/

[2012/11/20]

http://www.slideshare.net/eclaudioaugusto/curso-richfaces-333-i

http://www.slideshare.net/israelalcazar/introduccin-jsf-richfaces-e-icefaces-2267358

[2012/11/24]

4. ICEFACES

http://www.icesoft.org/wiki/display/ICE/Getting+Started

http://res.icesoft.org/docs/v1_8_2/components/index.html

[2012/12/210]

http://icefaces-showcase.icesoft.org/showcase.jsf?grp=compatMenu

http://www.icesoft.org/java/home.jsf

http://component-showcase.icesoft.org/component-showcase/

[2012/12/25]

5. JQUERY4JSF

https://code.google.com/p/jquery4jsf/

[2013/01/12]

6. PRIMEFACES

http://www.primefaces.org/showcase/ui/home.jsf

http://www.slideshare.net/gus_farfan/primefaces-14115155

http://www.kianworknotes.com/2013/04/primefaces-34-35-compatible-browsers.html

[2013/01/20]

http://www.adictosaltrabajo.com/tutorialesoriales.php?pagina=introduccionPrimefaces
http://www.mastertheboss.com/primefaces/primefaces-tutorial

http://chatejus.blogspot.com/2010/10/conozcamos-primefaces.html

http://aplicacioneslibres.bligoo.ec/media/users/19/951967/files/213572/PrimefacesJonathanGuerrero.pdf

[2013/02/10]

7. OPENFACES

http://openfaces.org/co mponents/

http://www.openfaces.org/documentation/developersGuide/

http://prezi.com/gxdbru3i0pii/openfaces/

http://www.openfaces.org/documentation/developersGuide/

http://www.jsfcentral.com/articles/openfaces_1.html

[2013/02/18]

8. MYFACES

http://myfaces.apache.org/

http://myfaces.apache.org/trinidad/browsers.html

http://myfaces.apache.org/tobago/demo.html

http://balteus.blogspot.com/2009/03/banco-de-experiencias-i-apache-myfaces.html

[2013/03/12]

9. ADF FACES

http://www.oracle.com/technetwork/developer-tools/adf/overview/index-092391.html

http://www.oracle.com/technetwork/developer-tools/adf/documentation/adf-faces-rc-demo-083799.html

https://forums.oracle.com/forums/thread.jspa?threadID=2367063

http://cafelojano.wordpress.com/2007/10/10/caracteristicas-de-adf-faces-11g-que-pueden-ahorrar-trabajo/

[2013/03/22]

10. RC FACES

http://www.rcfaces.org/starter/index.jsf

http://www.rcfaces.org/content/requirements.html

http://www.rcfaces.org/content/introduction.html

http://jdevadf.oracle.com/adf-richclient-demo/faces/index.jspx;jsessionid=2FKpRwNTnTRdXzwr6LHM8RhQchQbrG19Pnz8nqT2l3wJPYvcbVyL!-44105662

[2013/04/12]

11. PRETTYFACES

http://ocpsoft.org/prettyfaces/

http://stackoverflow.com/questions/11228336/prettyfaces-occasionally-uses-text-plain-instead-of-text-html

http://code.google.com/p/prettyfaces/issues/detail?id=124

[2013/04/20]