Comparativo de Frameworks web MVC open source en java, para la migración del administrador vortal en el CIADTI

Autor:

Oscar Alberto Bolaño Narvaez

Director:

Luis Alberto Esteban Villamizar

Magister en Informática

Universidad de Pamplona

Facultad de Ingenierías y Arquitectura

Departamento de Ingenierías Electrónica, Eléctrica, Sistemas y Telecomunicaciones

Programa de Ingeniería de Sistemas

Pamplona, Norte de Santander – Colombia

2017

Tabla de Contenidos

Índice de Figuras

Índice de Tablas

1 Introducción

Este capítulo contiene la información correspondiente a el problema, justificación y objetivos del proyecto, ofrece una explicación al lector del por qué la necesidad de desarrollar este proyecto de grado y el por qué, cómo, cuándo y dónde se llevará cabo.

1.1 Problema

Debido al constante crecimiento en el ámbito de las tecnologías enfocadas hacia la evolución del desarrollo web, se observa que la creación de sistemas software es indispensable para casi cualquier organización y si se mira el desarrollo tradicional de este tipo de sistemas, es indispensable la normalización de los datos sin importar la forma como se manipulen los mismos, el problema que se maneja desarrollando de manera tradicional es que un sistema es más que solo datos, se debe considerar también código fuente, librerías, archivos de configuración etc., y todo esto depende del individuo que esté desarrollando el sistema de información, lo cual, hace más complicado el proceso.

Un Framework web MVC es la solución a este inconveniente debido a que se encarga de automatizar muchos procesos en el desarrollo, sin embargo, la gran variedad de Frameworks web MVC existentes, se ha convertido en un factor de riesgo para los desarrollos en la medida que una mala selección, implica probablemente el fracaso del proyecto o en el mejor de los casos puede ser un obstáculo para el futuro mantenimiento de la aplicación por lo que el proceso de selección del Framework adecuado para el tipo de sistema a desarrollar se convierte en un factor determinante.

Es por esta razón, nace la necesidad de llevar a cabo este proyecto de grado para determinar cuál es el Framework web MVC que mejor se adapta al desarrollo de sistemas de

software en el Centro de Investigación Aplicada y Desarrollo en Tecnologías de Información CIADTI pensando en sostenibilidad, fácil mantenimiento y amigabilidad con el usuario final.

1.2 Justificación

El CIADTI ha desarrollado la infraestructura software de la Universidad de Pamplona desde hace aproximadamente más de 15 años lo que ha permitido acumular una gran cantidad de código bajo la misma arquitectura, que ha funcionado adecuadamente, pero que por el ritmo de crecimiento en las tecnologías de desarrollo podría en un futuro llegar a ser difícil de mantener.

Actualmente está realizando procesos de rediseño de interfaces de usuario para algunas de sus aplicaciones y es una oportunidad para incorporar un Framework web MVC adecuado y fácil de integrar con lo que actualmente cuenta.

Es así que un estudio de Frameworks web MVC es el primer paso para una adecuada selección y recomendación del más indicado en cuanto a facilidad de integración a los desarrollos con los que se dispone actualmente.

1.3 Objetivos

1.3.1 Objetivo general

Hacer un estudio comparativo entre diferentes Frameworks web MVC open source para la migración del administrador Vortal en el CIADTI.

1.3.2 Objetivos específicos

1. Realizar revisión bibliográfica de Frameworks web MVC open source que existen en la actualidad para el desarrollo web.

2. Identificar los dos mejores Frameworks web MVC open source en Java según características y arquitectura.
3. Establecer criterios para evaluar cada uno de los Frameworks seleccionados previamente y compararlos según resultados arrojados en cada criterio.
4. Seleccionar el mejor Framework según análisis realizado y ponerlo a prueba con el desarrollo de mínimo cinco funcionalidades de un prototipo de aplicación software acotado durante el proceso.
5. Analizar los resultados obtenidos en el desarrollo del prototipo.

1.4 Materiales

Para el desarrollo del proyecto se utilizaron diferentes materiales tangibles e intangibles como libros, software, equipos de cómputo a través de los cuales se pudo desarrollar el comparativo entre los Frameworks web MVC y selección del mejor, para ponerlo a prueba con las funcionalidades de un prototipo de software en el CIADTI. Los materiales utilizados en el desarrollo del proyecto se describen a continuación en las siguientes tablas.

Tabla 1: Materiales Software

#	Nombre	Descripción
1	Administrador Vortal	Utilizado para analizar el funcionamiento actual e identificar posibles mejoras.
2	NetBeans 8.1	Utilizado para el desarrollo del prototipo.
3	JavaServer Faces	Framework utilizado para el desarrollo del prototipo.

4	Postgres 9.2	Utilizado como motor de base de datos para el prototipo.
5	pgAdmin III	Utilizado como gestor de base de datos del prototipo.

Tabla 2: Materiales Hardware

#	Nombre	Descripción
1	Equipo pc de escritorio	Utilizado para investigar, desarrollar el documento escrito y el prototipo

Tabla 3: Materiales Talento humano

#	Nombre	Descripción
1	Personal de desarrollo tecnológico	Recopilación de información con respecto a funcionamiento de base de datos y aplicativo actual.

1.5 Metodología

1.5.1 Paradigma

La investigación tiene un enfoque cualitativo donde se persiguió la selección del Framework más apropiado para la transformación del administrador Vortal en el CIADTI y la construcción de un prototipo del mismo, previamente haciendo un análisis comparativo entre los Frameworks web MVC open source más conocidos en Java.

1.5.2 Alcance

Tiene un alcance exploratorio en el que se intentó desarrollar el diagnóstico del estado actual del aplicativo administrador Vortal en el CIADTI, seguidamente se identificó el Framework más apropiado para proponer la migración del mismo.

1.5.3 Tipo

Su diseño es no experimental ya que no se hizo ningún tipo de experimento dentro del proceso de investigación. La población objetivo es de 30.000 aproximadamente, esta población es la afectada por la transformación del administrador Vortal en el CIADTI, incluye estudiantes, docentes, administrativos y personal del CIADTI.

1.5.4 Fuentes de información

Fue oportuno investigar en fuentes virtuales, en las cuales se hizo uso de archivos PDF y sitios oficiales de los Frameworks, todas estas fuentes confiables y oportunas para el proceso de investigación, además se contó con la información proporcionada por el grupo de desarrollo tecnológico en el CIADTI.

1.5.5 Población

El proyecto va dirigido principalmente al CIADTI de la Universidad de Pamplona pero teniendo en cuenta que el aplicativo administrador Vortal es utilizado por todos los estudiantes, administrativos y docentes también se ven directamente afectados.

1.5.6 Instrumentos

Como principal instrumento para recolección de información además de los ya mencionados anteriormente, fue el seguimiento al funcionamiento del actual aplicativo administrador Vortal y algunos procesos que se llevan a cabo en el área de desarrollo tecnológico del CIADTI para su mantenimiento, así se pudo ir documentando el proceso y adecuando con mejoras incluidas.

1.5.7 Consideraciones éticas

Entre las consideraciones éticas se tuvo en cuenta que toda la información correspondiente al proceso de selección del Framework web MVC open source más apropiado para la transformación del administrador Vortal en el CIADTI fue investigada en sus sitios oficiales y en este proyecto se contó con acceso a información de la base de datos de prueba del CIADTI con la mayor responsabilidad necesaria, ya que esta información les pertenece y no puede ser tomada para otros fines.

2 Marco teórico y estado del arte

Este capítulo contiene información correspondiente a las teorías relacionadas con los temas tratados en este proyecto con el fin de esclarecer dudas a los lectores, además, muestra en forma cronológica investigaciones y estudios que se hayan hecho en el mundo con relación a Frameworks web MVC open source.

2.1 Ingeniería del Software

Es una disciplina de la ingeniería que comprende todos los aspectos de la producción de software desde las etapas iníciales de la especificación del sistema, hasta el mantenimiento de éste después que se utiliza. En esta definición existen dos frases clave:

Disciplina de la ingeniería. Los ingenieros hacen que las cosas funcionen. Aplican teorías, métodos, herramientas donde sean convenientes, pero las utilizan de forma selectiva y siempre tratando de descubrir soluciones a los problemas, aun cuando no existan teorías o métodos para resolverlos.

Todos los aspectos de producción de software. La ingeniería del software no solo comprende los procesos técnicos del desarrollo de software, sino también con actividades tales como la gestión de proyectos de software y el desarrollo de herramientas, métodos y teorías de apoyo a la producción de software (Sommerville, 2005).

En general, los ingenieros de software adoptan un enfoque sistemático y organizado en su trabajo, ya que es la forma más efectiva de producir software de alta calidad. Sin embargo, aunque la ingeniería consiste en adoptar el método más apropiado para un conjunto de circunstancias, un enfoque más informal y creativo de desarrollo podría ser efectivo en algunas

circunstancias, el desarrollo informal es apropiado para el desarrollo de sistemas basados en web, los cuales requieren una mezcla de técnicas de software y diseño gráfico (Sommerville, 2005).

2.2 Software libre vs open source

2.2.1 Software libre

El «software libre» es una cuestión de libertad, no de precio. Para comprender este concepto, se debe pensar en la acepción de libre como en «libertad de expresión» y no como en «barra libre de cerveza». Con software libre se hace referencia a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. Se refiere a cuatro clases de libertad para los usuarios de software:

Libertad 0: la libertad para ejecutar el programa sea cual sea su propósito.

Libertad 1: la libertad para estudiar el funcionamiento del programa y adaptarlo a sus necesidades —el acceso al código fuente es condición indispensable para esto.

Libertad 2: la libertad para redistribuir copias y ayudar así al vecino.

Libertad 3: la libertad para mejorar el programa y luego publicarlo para el bien de toda la comunidad —el acceso al código fuente es condición indispensable para esto.

Software libre es cualquier programa cuyos usuarios gocen de estas libertades. De modo que se debería ser libre de redistribuir copias con o sin modificaciones, de forma gratuita o cobrando por su distribución, a cualquiera y en cualquier lugar. Gozar de esta libertad significa, entre otras cosas, no tener que pedir permiso ni pagar para ello.

Asimismo, deberías ser libre para introducir modificaciones y utilizarlas de forma privada, ya sea en tu trabajo o en tu tiempo libre, sin siquiera tener que mencionar su existencia. Si decidieras publicar estos cambios, no deberías estar obligado a notificárselo a ninguna persona ni de ninguna forma en particular.

La libertad para utilizar un programa significa que cualquier individuo u organización podrán ejecutarlo desde cualquier sistema informático, con cualquier fin y sin la obligación de comunicárselo subsiguientemente ni al desarrollador ni a ninguna entidad en concreto.

La libertad para redistribuir copias supone incluir las formas binarias o ejecutables del programa y el código fuente tanto de las versiones modificadas como de las originales —la distribución de programas en formato ejecutable es necesaria para su adecuada instalación en sistemas operativos libres. No pasa nada si no se puede producir una forma ejecutable o binaria —dado que no todos los lenguajes pueden soportarlo—, pero todos deben tener la libertad para redistribuir tales formas si se encuentra el modo de hacerlo.

Para que las libertades 2 y 4 —la libertad para hacer cambios y para publicar las versiones mejoradas— adquieran significado, se debe disponer del código fuente del programa. Por consiguiente, la accesibilidad del código fuente es una condición necesaria para el software libre.

Para materializar estas libertades, deberán ser irrevocables siempre que no se comente ningún error; si el desarrollador del software pudiera revocar la licencia sin motivo, ese software dejaría de ser libre.

Sin embargo, ciertas normas sobre la distribución de software libre pueden parecer aceptables siempre que no planteen un conflicto con las libertades centrales. Por ejemplo, el

copyleft, grosso modo, es la norma que establece que, al redistribuir el programa, no pueden añadirse restricciones que nieguen a los demás sus libertades centrales. Esta norma no viola dichas libertades, sino que las protege.

De modo que puedes pagar o no por obtener copias de software libre, pero independientemente de la manera en que las obtengas, siempre se tendrá libertad para copiar, modificar e incluso vender estas copias.

El software libre no significa que sea «no comercial». Cualquier programa libre estará disponible para su uso, desarrollo y distribución comercial. El desarrollo comercial del software libre ha dejado de ser excepcional y de hecho ese software libre comercial es muy importante.

Las normas sobre el empaquetamiento de una versión modificada son perfectamente aceptables siempre que no restrinjan efectivamente tu libertad para publicar versiones modificadas. Por la misma razón, serán igualmente aceptables aquellas normas que establezcan que «si distribuyo el programa de esta forma, deberás distribuirlo de la misma manera» —cabe destacar que esta norma te permite decidir si publicar o no el programa. También se admite la posibilidad de que una licencia exija enviar una copia modificada y distribuida de un programa a su desarrollador original.

En el proyecto GNU, se utiliza el «copyleft» para proteger legalmente estas libertades. Pero también existe software libre sin copyleft. Se cree que hay razones de peso para recurrir al copyleft, pero si tu programa, software libre, carece de él, todavía se tendrá la opción de seguir utilizándolo.

A veces la normativa gubernamental de control de las exportaciones y las sanciones comerciales pueden constreñir tu libertad para distribuir copias a nivel internacional. Los

desarrolladores de software no tienen el poder para eliminar o invalidar estas restricciones, pero lo que sí pueden y deben hacer es negarse a imponer estas condiciones de uso al programa. De este modo, las restricciones no afectarán a las actividades y a los individuos fuera de la jurisdicción de estos gobiernos.

Cuando se habla de software libre, es preferible evitar expresiones como «regalar» o «gratis», porque entonces se caerá en el error de interpretarlo como una mera cuestión de precio y no de libertad (Stallman, 2004).

2.2.2 Open source

(Código abierto) es el término con el que se conoce al software distribuido y desarrollado libremente. El código abierto tiene un punto de vista más orientado a los beneficios prácticos de compartir el código (GPSOS, 2017).

El código abierto no significa simplemente acceso al código fuente. Los términos de distribución del software de fuente abierta deben cumplir con los siguientes criterios (OpenSource Initiative, 2007):

Libre redistribución. La licencia no debe restringir a nadie vender o entregar el software como un componente de una distribución de software que contenga programas de distintas fuentes. La licencia no debe requerir ningún tipo de cuota por su venta.

Código fuente. El programa debe incluir el código fuente, y se debe permitir su distribución tanto como código fuente como compilado. Cuando de algún modo no se distribuya el código fuente junto con el producto, deberá proveerse un medio conocido para obtener el código fuente sin cargo, a través de Internet. El código fuente es la forma preferida en la cual un programador modificará el programa. No se permite el código fuente deliberadamente

confundido (obfuscation). Tampoco se permiten formatos intermedios, como la salida de un preprocesador, o de un traductor.

Trabajos derivados. La licencia debe permitir modificaciones y trabajos derivados, y debe permitir que estos se distribuyan bajo las mismas condiciones de la licencia del software original.

Integridad del código fuente del autor. La licencia puede restringir la distribución de código fuente modificado sólo si se permite la distribución de "patch files" con el código fuente con el propósito de modificar el programa en tiempo de construcción. La licencia debe permitir explícitamente la distribución de software construido en base a código fuente modificado. La licencia puede requerir que los trabajos derivados lleven un nombre o número de versión distintos a los del software original.

No discriminar personas o grupos. La licencia no debe hacer discriminación de personas o grupos de personas.

No discriminar campos de aplicación. La licencia no debe restringir el uso del programa en un campo específico de aplicación. Por ejemplo, no puede restringir su uso en negocios, o en investigación genética.

Distribución de la licencia. Los derechos concedidos deben ser aplicados a todas las personas a quienes se redistribuya el programa, sin necesidad de obtener una licencia adicional.

La licencia no debe ser específica a un producto. Los derechos aplicados a un programa no deben depender de la distribución particular de software de la que forma parte. Si el programa es extraído de esa distribución y usado o distribuido dentro de las condiciones de la licencia del

programa, todas las personas a las que el programa se redistribuya deben tener los mismos derechos que los concedidos en conjunción con la distribución original de software.

La licencia no debe contaminar otro software. La licencia no debe imponer restricciones sobre otro software que es distribuido junto con él. Por ejemplo, la licencia no debe insistir en que todos los demás programas distribuidos en el mismo medio deben ser software open-source.

La licencia debe ser tecnología neutral. Ninguna disposición de la licencia puede basarse en cualquier tecnología individual o estilo de interfaz.

Algunos ejemplos de este tipo de licencia son: GNU GPL, X Consortium, Artistic, MPL, entre otras.

2.2.3 Diferencias entre software libre y open source

En la práctica, el código abierto sostiene criterios menos estrictos que los del software libre. Por lo que se sabe todo el código fuente de software libre existente que se ha publicado se podría calificar como código abierto. Casi todo el software de código abierto es software libre, con algunas excepciones. En primer lugar, algunas licencias de código abierto son demasiado restrictivas, por lo que no se las puede considerar como libres. Por ejemplo, «Open Watcom» no es libre porque su licencia no permite hacer versiones modificadas y utilizarlas de forma privada. Afortunadamente, son muy pocos los programas que llevan tales licencias.

En segundo lugar, y lo que en la práctica es más importante, muchos productos que funcionan como ordenadores verifican las firmas de sus programas ejecutables para impedir que los usuarios instalen ejecutables diferentes; solo una compañía tiene el privilegio de elaborar ejecutables que funcionen en el dispositivo y de acceder a todas las prestaciones del mismo. A estos dispositivos se es llama «tiranos» y la práctica se denomina «tivoización», por referencia al

producto (Tivo) donde por primera vez se descubrió su implementación. Aun cuando el ejecutable esté hecho a partir de código fuente libre, los usuarios no pueden ejecutar versiones modificadas, de modo que el ejecutable no es libre.

Los criterios del código abierto no contemplan esta cuestión, solo les interesa la licencia del código fuente. De modo que estos ejecutables no modificables, si están hechos a partir de un código fuente como Linux, que es de código abierto y libre, son de código abierto pero no son libres. Muchos productos Android contienen ejecutables tivoizados de Linux, que no son libres (Stallman Richard, 2016)

2.3 Modelo Vista Controlador (MVC)

El patrón Modelo-Vista-Controlador (MVC) surge con el objetivo de reducir el esfuerzo de programación, necesario en la implementación de sistemas múltiples y sincronizados de los mismos datos, a partir de estandarizar el diseño de las aplicaciones. El patrón MVC es un paradigma que divide las partes que conforman una aplicación en el Modelo, las Vistas y los Controladores, permitiendo la implementación por separado de cada elemento, garantizando así la actualización y mantenimiento del software de forma sencilla y en un reducido espacio de tiempo. A partir del uso de Frameworks basados en el patrón MVC se puede lograr una mejor organización del trabajo y mayor especialización de los desarrolladores y diseñadores (Fernández & Yanette, 2012).

Separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la

interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento (Anonimo, 2017).

2.3.1 Historia del MVC

El patrón MVC fue una de las primeras ideas en el campo de las interfaces gráficas de usuario y uno de los primeros trabajos en describir e implementar aplicaciones software en términos de sus diferentes funciones.

MVC fue introducido por Trygve Reenskaug (web personal) en Smalltalk-76 durante su visita a Xerox Parc en los años 70 y, seguidamente, en los años 80, Jim Althoff y otros implementaron una versión de MVC para la biblioteca de clases de Smalltalk-80. Solo más tarde, en 1988, MVC se expresó como un concepto general en un artículo (Microsoft, 2016) sobre Smalltalk-80.

En esta primera definición de MVC el controlador se definía como el módulo que se ocupa de la entrada (de forma similar a como la vista se ocupa de la salida). Esta definición no tiene cabida en las aplicaciones modernas en las que esta funcionalidad es asumida por una combinación de la 'vista' y algún Framework moderno para desarrollo. El 'controlador', en las aplicaciones modernas de la década de 2000, es un módulo o una sección intermedia de código, que hace de intermediario de la comunicación entre el 'modelo' y la 'vista', y unifica la validación (utilizando llamadas directas o el *server* para desacoplar el 'modelo' de la 'vista' en el 'modelo' activo).

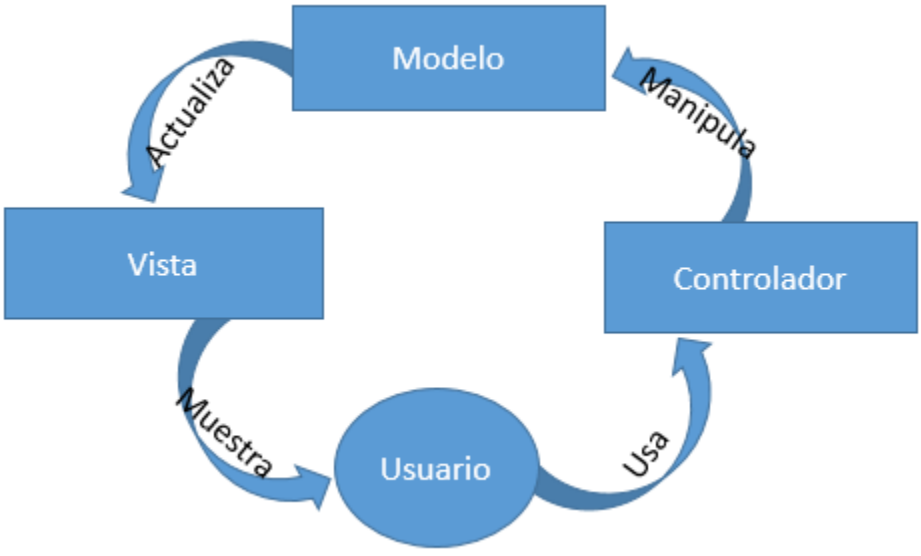
Algunos aspectos del patrón MVC han evolucionado dando lugar a ciertas variantes del concepto original, ya que las partes del MVC clásico realmente no tienen sentido para los clientes actuales:

* HMVC (MVC Jerárquico)
* MVA (Modelo-Vista-Adaptador)
* MVP (Modelo-Vista-Presentador)
* MVVM (Modelo-Vista Vista-Modelo)
* ... y otros que han adaptado MVC a diferentes contextos.

2.3.2 Descripción de MVC

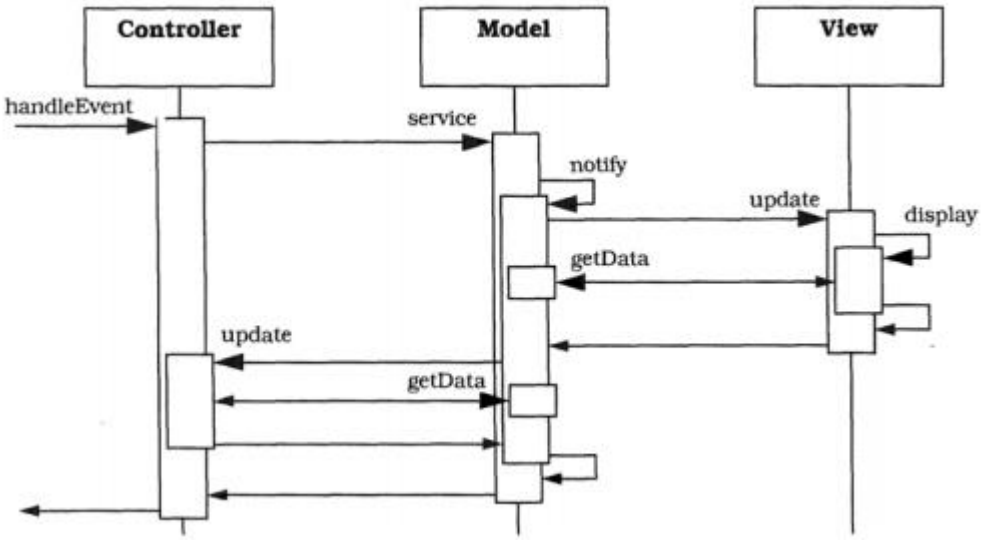
La gráfica muestra la estructura del modelo vista controlador, sus diferentes componentes y cómo se colaboran entre ellos funcionando como un sistema.

Figura 1. Esquema MVC



Fuente:<http://material.concursos.econo.unlp.edu.ar/concursos/T%C3%A9cnico-Profesional%20(Inform%C3%A1tica)/patrones/Modelo%E2%80%93vista%E2%80%93controlador.pdf> - [citado el 20 de febrero de 2017]

Figura 2. Patrón MVC



Fuente: Javier J. Gutiérrez. ¿Qué es un Framework web? - [En línea] - <http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf> [citado el 20 de febrero de 2017]

Los componentes del MVC se podrían definir de la siguiente manera:

El Modelo: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.

El Controlador: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta de 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos),

por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo' (véase Middleware).

La Vista: Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho 'modelo' la información que debe representar como salida.

2.3.3 Interacción entre componentes

Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo de control que se sigue generalmente es el siguiente:

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.).
2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.
3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.
4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se reflejan los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, se podría utilizar el patrón Observador para proveer cierta indirección

entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. Este uso del patrón Observador no es posible en las aplicaciones web puesto que las clases de la vista están desconectadas del modelo y del controlador. En general el controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice.

Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista. Por ejemplo en el MVC usado por Apple en su framework Cocoa. Suele citarse como Modelo-Interface-Control, una variación del MVC más puro.

5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente...(Anonimo, 2017).

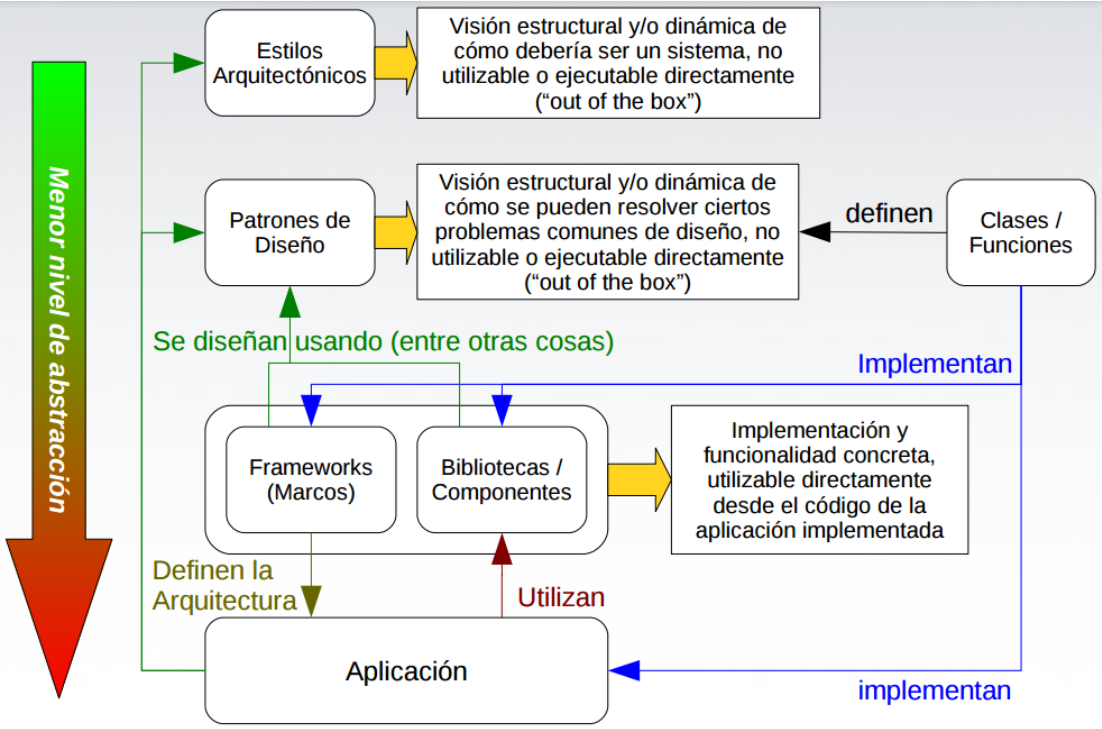
2.4 Framework web

Un Framework (armazón), es una abstracción en la que cierto código común provee una funcionalidad genérica que puede ser sobrescrita o especializada de forma selectiva por medio de código con funcionalidad específica provisto por los clientes del Framework (desarrolladores de software / programadores) (Gutierrez, 2010)

El concepto Framework se emplea en muchos ámbitos del desarrollo de sistemas software, no solo en el ámbito de aplicaciones web. Se Puede encontrar Frameworks para el desarrollo de aplicaciones médicas, de visión por computador, para el desarrollo de juegos, y para cualquier ámbito que pueda ocurrírsenos.

En general, con el término Framework, se hace referencia a una estructura software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un Framework se puede considerar como una aplicación genérica incompleta y configurable a la que se puede añadir las últimas piezas para construir una aplicación concreta.

Figura 3. Arquitectura de software y Frameworks web



Fuente: Demián Gutierrez - Universidad de los Andes Venezuela – [en línea] <http://www.codecompiling.net/files/slides/IS_clase_10_frameworks_componentes.pdf> - [citado el 21 de febrero de 2017]

Los objetivos principales que persigue un Framework son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones. Un Framework web, por tanto, se puede definir como un conjunto de componentes

(por ejemplo clases y descriptores y archivos de configuración en XML) que componen un diseño reutilizable que facilita y agiliza el desarrollo de sistemas web (Javier, 2015)

Un Framework facilita el desarrollo de software permitiendo a los diseñadores y programadores dedicar su tiempo a lograr los requerimientos de software en lugar de lidiar con los detalles de bajo nivel necesarios para obtener un sistema funcional.

Por ejemplo, un equipo que está desarrollando un sistema web bancario al usar un Framework de desarrollo web puede enfocarse en el desarrollo de las operaciones de retiro y transferencias de dinero en lugar de tener que enfocarse en la mecánica del manejo de las peticiones HTTP o el manejo de las sesiones de los usuarios y el estado de la aplicación.

2.4.1 Tipos de Framework web

Según su aplicabilidad se dividen en dos tipos:

Frameworks de aplicación: Proveen la funcionalidad básica de una aplicación tratándose generalmente de interfaces gráficas y lógica de administración del sistema. A partir de ´este, se crea cada aplicación especializada. Sin embargo, estos Frameworks no contienen elementos del dominio mismo de las aplicaciones.

Frameworks de dominio específico: Modelan los objetos de un dominio específico y proveen la lógica genérica de una aplicación en este dominio. De este modo, una aplicación puede ser creada configurando los objetos del dominio y extendiendo la lógica de aplicación genérica. Por ser muy específicos y requerir un conocimiento muy preciso de los dominios son los más costosos de desarrollar.

Según las características de personalización los tipos de instanciación los Frameworks se pueden clasificar en tres categorías:

Caja blanca: Tanto los desarrolladores de aplicaciones (DA) como los desarrolladores de componentes (DC) necesitan conocer la arquitectura para adaptarla a una aplicación concreta. Los puntos de entrada se presentan como clases y métodos abstractos que deben ser implementados para la extensión del Framework, se tiene acceso al código fuente y se permite reutilizar la funcionalidad encapsulada en sus clases mediante herencia y reescritura de métodos.

Caja negra: Ocultan su estructura interna, los usuarios solo conocen una descripción general del Framework y sus puntos flexibles. Se extienden mediante composición y delegación. El Framework tiene definido una serie de interfaces que deben implementar los componentes que extienden el Framework.

Caja gris: Define una forma intermedia de reutilización, el Framework presenta características tanto de caja blanca como de caja negra (Pérez, 2013)

2.4.2 Características

En la siguiente tabla se pueden evidenciar algunas características que poseen casi todos los Frameworks web MVC.

Tabla 4: Características de los Frameworks web

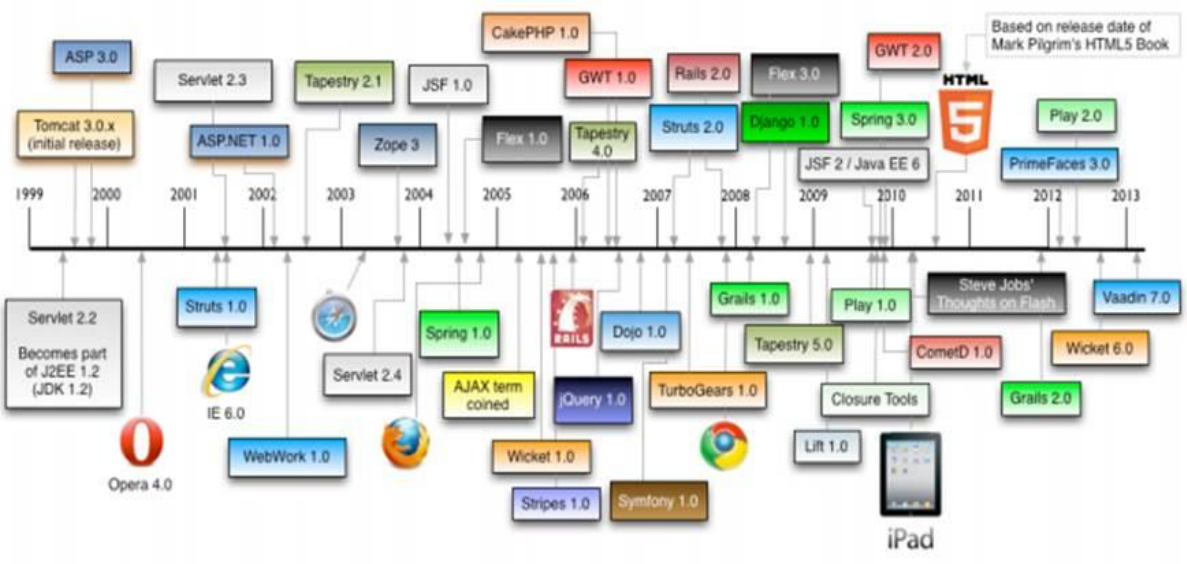
Abstracción de URLs y sesiones.	No es necesario manipular directamente las URLs ni las sesiones, el Framework ya se encarga de hacerlo.
Acceso a datos.	Incluyen las herramientas e interfaces necesarias para integrarse con herramientas de acceso a datos, en BBDD, XML, etc…
Controladores.	La mayoría de Frameworks implementa una serie de controladores para gestionar eventos, como una introducción de datos mediante un formulario o el acceso a una página. Estos controladores suelen ser fácilmente adaptables a las necesidades de un proyecto concreto.
Autentificación y control de acceso.	Incluyen mecanismos para la

	identificación de usuarios mediante login y password y permiten restringir el acceso a determinas páginas a determinados usuarios.
Internacionalización.	
Separación entre diseño y contenido.	

2.5 Estado del arte

Existe una gran cantidad de Frameworks web MVC en el mundo la siguiente gráfica describe la evolución de algunos de los más conocidos.

Figura 4. Evolución en el tiempo de Frameworks web.



Fuente: Raible Designs – comparing jvm web frameworks [en línea] - <http://static.raibledesigns.com/repository/presentations/Comparing_JVM_Web_Frameworks_February2014.pdf> [citado el 20 de febrero de 2017]

Para analizar la evolución y comportamiento actual de Frameworks web se hará una división de los que existen en Java y en PHP, dos grandes pilares del desarrollo web.

2.5.1 Frameworks web MVC open source en Java

2.5.1.1 Orígenes de los Frameworks para aplicaciones web en Java.

El surgimiento de los servlets de Java resultó en un avance bastante productivo en relación con el estándar CGI, ya que resultaban más poderosos y rápidos, así como

portables y fácilmente extensibles; sin embargo, el desplegar código HTML en el explorador a través del método println(), especialmente cuando se trataba de gran cantidad de líneas, resultaba problemático y agotador.

Ante esta situación, llegaron al mundo los JSP (JavaServer Pages), invirtiendo el concepto de los servlets, permitiendo insertar fácilmente código Java dentro de la página HTML. Con esta solución, las aplicaciones web adoptaron a los JSP como figura central, lo que pronto traería como consecuencia problemas en el control del flujo, así como en el mantenimiento de páginas con demasiado código Java.

Ante esta nueva situación, se llegó a la idea de utilizar ambas tecnologías de manera conjunta, lo que representó una buena opción y satisfizo las necesidades de los desarrolladores, aunque sólo por un tiempo, ya que con la experimentación e implementación de dichas tecnologías, que además se presentaban como estándares, la comunidad que las había utilizado llegó a la conclusión de que lo que se les ofrecía, o bien no era suficiente para cumplir con los requerimientos de los diferentes proyectos, o lo realizaba de una manera no óptima o por debajo del nivel requerido.

Esto condujo a que los programadores desarrollaran sus propios medios para cumplir con sus necesidades, lo que con el tiempo traería como resultado la aparición de los primeros Frameworks orientados a las aplicaciones web.

2.5.1.2 Struts

Historia.

Es un Framework de código abierto usado para desarrollar aplicaciones web. Fue originalmente desarrollado por Craig R. McClanahan, y en el año 2002 pasa a pertenecer a la Apache Software Foundation.

Struts provee un Framework que facilita el desarrollo organizando los elementos J2EE: JSP, clases, HTMLs, etc, consiguiendo convertirse en un referente de los demás Frameworks, integrándose de una forma efectiva con las tecnologías Java disponibles en el momento.

Struts se define como un Framework basado en la arquitectura MVC, la cual articula los desarrollos definiendo claramente los roles de los elementos que se desarrollan. Así se encuentra el modelo que gestiona la lógica de negocios y el acceso a datos. La vista que es el responsable de la percepción final del usuario. Por último el Controlador que es responsable de los aspectos navegacionales y de coordinación.

Los motivos por los que surge Struts2 hay que buscarlos en limitaciones de Struts, ya que la versión 1 plantea restricciones en la forma de desarrollar, la aparición de nuevos Frameworks que aportan nuevos conceptos y puntos de vistas, y el surgimiento de nuevas tecnologías como Ajax.

El desarrollo de Struts 2 está pensado para mejorar Struts aportando flexibilidad, sencillez y robustez a los elementos que se emplean en el desarrollo, facilitar el desarrollo de nuevas aplicaciones y el mantenimiento de aquellas que se desarrollen. Por ello se hace una nueva arquitectura, una nueva API, nuevos Tags, etc., a partir de un Framework existente WebWork, pero manteniendo todo lo bueno que tenía Struts. El resultado es un Framework sencillo y fiable.

Versiones.

Estas son algunas de las versiones distribuidas para Struts 1, desde el lanzamiento, en el año 2000:

- 2002-02-10: Struts 1.0.2
- 2002-03-20: Struts 1.1 Beta 1
- 2002-08-13: Struts 1.1 Beta 2
- 2002-12-31: Struts 1.1 Beta 3
- 2003-02-22: Struts-1.1-rc1
- 2003-06-10: Struts-1.1-rc2
- 2003-06-30: Struts-1.1
- 2004-08-27: Struts-1.2.2
- 2004-09-13: Struts-1.2.4
- 2004-11-20: Struts-1.2.6
- 2005-05-06: Struts-1.2.7
- 2008-12-01: Struts-1.3.10 (última distribución por EOL)
- 2016-05-09: Struct-2.5 (versión actual)

Características.

Diseño Simplificado: Uno de los principales problemas que tenía el Framework Struts 1 era el uso de clases abstractas, cosa que cambia en la versión 2 en la cual se hace uso de Interfaces. Esto le provee una mayor facilidad para la extensión y la adaptación, ya que los interfaces son más fáciles de adaptar que las clases abstractas. Otro cambio es que se busca que las clases que sean lo más simple posible con lo que los actions se convierten en POJOs, elementos que además estarán poco acoplados. Los POJOs son clases que cuentan con getter y setter para poder recibir valores desde páginas, y cuentan con algunos métodos en los cuáles se pondrá la lógica de negocio.

Simplificación de los actions: Como se ha dicho los actions son POJOs. Cualquier clase java con un método execute puede actuar como un Action. Así no se hace necesario implementar ningún interfaz. Además se introduce la Inversión de control en la gestión de los actions.

Desaparecen los ActionForms: se ven reemplazados por simples Java Beans que son usados para leer las propiedades directamente. Lo usual es que el propio Action actúe de JavaBean, con lo que se facilita el desarrollo. Además ha mejorado la lectura de parámetros con el objetivo de no tener únicamente propiedades de tipo String.

Test simplificados: como los actions engloban la lógica de negocio y los JavaBeans, es más sencillo hacer test unitarios.

Fácil selección de opciones por defecto: casi todos los elementos de configuración tienen definidos un valor por defecto que se puede parametrizar, lo que facilita la elección de acciones por defecto.

Results mejorados: a diferencia de los Action Forwards, los results de Struts 2 son más flexibles a la hora de poder definir múltiples elementos de la vista. Además desaparece la complejidad de utilizar ActionForwards, ya que se sustituye por la devolución de Strings.

Mejoras en Tags: Struts 2 permite añadir capacidades utilizando tags que permite hacer páginas consistentes sin añadir código. Los tags presentan más opciones, están orientados a los results y pueden ser cambiados de forma sencilla. Además se añaden tags de marcado (markup) los cuáles son editables usando templates FreeMarker. Esto significa que se puede hacer que un mismo tag se comporte de forma diferente sin tener que hacer ninguna tarea de programación.

Se introducen anotaciones: las aplicaciones en struts 2 puede usar anotaciones como alternativa a XML y configuraciones basadas en properties.

Arranque rápido: muchos cambios se pueden hacer sin necesidad de reiniciar el contenedor web.

Parametrización del controlador: Struts 1 permitía parametrizar el procesador de peticiones a nivel de módulo. Struts 2 lo permite a nivel de action.

Fácil integración con Spring.

Facilidad para añadir plugins.

Soporte de Ajax: se incluye un theme AJAX que permite hacer aplicaciones interactivas de forma más sencilla (sicrea.com.mx, 2009)

Arquitectura.

Un usuario envía una petición: Un usuario realiza la petición de un recurso dentro del servidor.

El elemento FilterDispatcher determina la acción que deberá responder: El Framework dispone de los elementos requeridos para que el dispatcher sea capaz de determinar qué action es el responsable de recibir la petición y procesarla. Para ello se apoya en el framework para la publicación del recurso, y para su ejecución.

Se aplican los interceptores definidos: Existen diferentes interceptores que se pueden configurar para que ejecuten diferentes funcionalidades como workflows, validaciones, upload de ficheros, etc.

Se ejecuta el Action: Tras la ejecución de los diferentes interceptores el método específico del action es ejecutado, realizándose aquellas operaciones y acciones que se

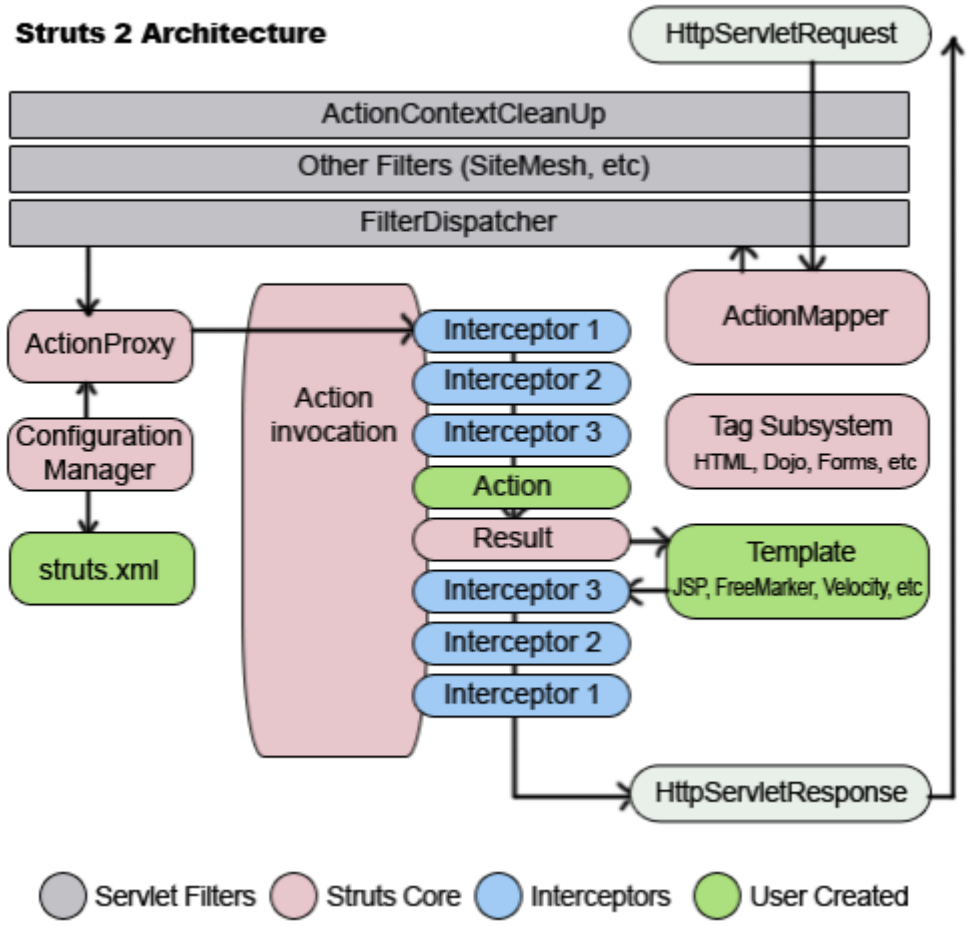
hayan definido. El action termina devolviendo un resultado el cuál se utiliza para determinar la página a devolver.

Se renderiza la salida: Tras la ejecución del action se determina cuál es la página que se devuelve y se ejecutan el forward a dicha página.

Se devuelve la petición: Para realizar la devolución se ejecutan los interceptores que correspondan y se procede a devolver la petición al cliente. De esta forma es posible añadir lógica externa a los servidores también en la devolución.

Se muestra el resultado al cliente final: Finalmente el control es devuelto al cliente quien podrá visualizar el resultado en su navegador.13

Figura 5. Arquitectura del Framework web Struts



Fuente: Manual de Struts2. [En línea] - <http://www.sicrea.com.mx/media/archivos/ManualStruts2Esp anol.pdf> - [citado el 21 de febrero de 2017]

2.5.1.3 Java Server Faces (JSF)

Historia.

En mayo de 2001 varias proveedores de software libre incluido Sun, Oracle, IBM Y BEA a través del Java Community process, consideran favorable continuar con el desarrollo de la especificación completa y detallada para crear aplicaciones Java EE, el principal objetivo de esta era proporcionar un estándar simple para construir User

interface (UI) para aplicaciones Java web. Esto dio lugar al Java Specification Request (JSR) y a partir de este nació Java Server Faces (Hernán, 2013)

JavaServer Faces (JSF) es el Framework para aplicaciones web en Java de Sun Microsystems, liberado apenas en Marzo del 2004, que busca tomar su lugar como estándar entre los muchos de su clase.

JSF es un Framework orientado a la interfaz gráfica de usuario (GUI), facilitando el desarrollo de éstas, y que sin embargo, realiza una separación entre comportamiento y presentación, además de proporcionar su propio servlet como controlador, implementando así los principios del patrón de diseño Model-View-Controller (MVC), lo que da como resultado un desarrollo más simple y una aplicación mejor estructurada.

El enfoque mencionado anteriormente no es nada nuevo. Lo que hace a JSF tan atractivo, entre muchas otras cosas más, es que brinda un modelo basado en componentes y dirigido por eventos para el desarrollo de aplicaciones web, que es similar al modelo usado en aplicaciones GUI standalone durante años [Bergsten, 2004], como es el caso de Swing, el framework estándar para interfaces gráficas de Java (JavaServer Faces, 2010).

Versiones.

- JSF 1.0 (11-03-2004) - Lanzamiento inicial de las especificaciones de JSF.
- JSF 1.1 (27-05-2004) - Lanzamiento que solucionaba errores. Sin cambios en las especificaciones ni en el renderkit de HTML.
- JSF 1.2 (11-05-2006) - Lanzamiento con mejoras y corrección de errores.
- JSF 2.0 (12-08-2009) - Lanzamiento con mejoras de funcionalidad, rendimiento y facilidad de uso.

- JSF 2.1 (22-10-2010) - Lanzamiento de mantenimiento, con mínimos cambios.
- JSF 2.2 (16-04-2013) - Lanzamiento que introduce soporte a HTML 5, Faces Flow, Stateless views y Resource library contracts.}
- JSF 2.2.13 (04-02-2016) – Mojarra. Versión actual.

Características.

1. Utiliza facelets para generar las vistas, añadiendo una biblioteca de etiquetas propia para crear los elementos de los formularios HTML.
2. Asocia a cada vista con formularios un conjunto de objetos java manejados por el controlador (managed beans) que facilitan la recogida, manipulación y visualización de los valores mostrados en los diferentes elementos de los formularios.
3. Introduce una serie de etapas en el procesamiento de la petición, como por ejemplo la de validación, reconstrucción de la vista, recuperación de los valores de los elementos, etc.
4. Utiliza un sencillo fichero de configuración para el controlador en formato xml.
5. Es extensible, pudiendo crearse nuevos elementos de la interfaz o modificar los ya existentes.
6. Forma parte del estándar J2EE. En efecto, hay muchas alternativas para crear la capa de presentación y control de una aplicación web java, como Struts y otros Frameworks, pero solo JSP forma parte del estándar.

Arquitectura.

Una aplicación JSF es similar a cualquier otra aplicación web basada en tecnología Java; Se ejecuta en un contenedor de servlets de Java y contiene:

Componentes de JavaBeans como modelos que contienen funcionalidad y datos específicos de la aplicación.

Una biblioteca de etiquetas personalizadas para representar manejadores de eventos y validadores.

Una biblioteca de etiquetas personalizadas para renderizar componentes de interfaz de usuario.

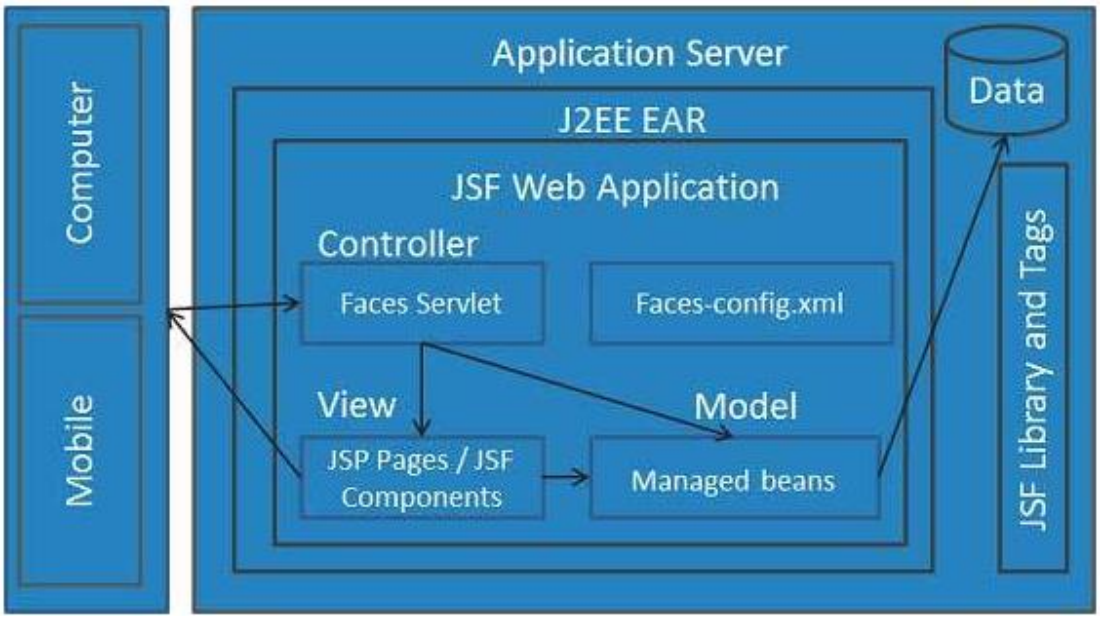
Componentes de la interfaz de usuario representados como objetos con estado en el servidor.

Clases de ayuda del servidor

Validadores, controladores de eventos y controladores de navegación

Archivo de recursos de configuración de aplicaciones para configurar recursos de aplicaciones

Figura 6. Arquitectura del Framework web Struts



Fuente: JSF – Architecture. [en línea] - < https://www.tutorialspoint.com/jsf/jsf_architecture.htm> - [citado el 22 de febrero de 2017]

Existen controladores que se pueden utilizar para realizar acciones de usuario. UI puede ser creado por autores de páginas web y la lógica empresarial puede ser utilizada por beans administrados.

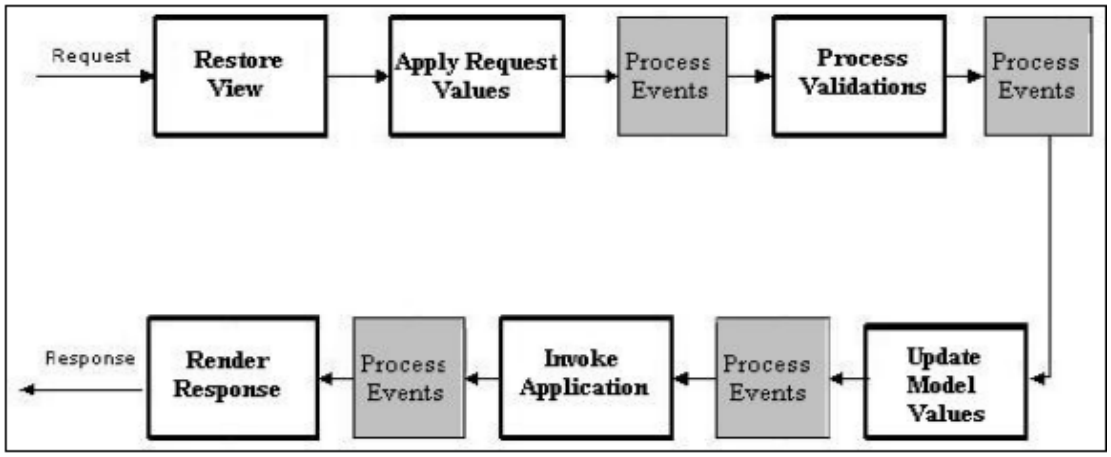
JSF proporciona varios mecanismos para representar un componente individual. Es responsabilidad del diseñador de páginas web seleccionar la representación deseada y el desarrollador de la aplicación no necesita saber qué mecanismo se utilizó para representar un componente de interfaz de usuario JSF (TutorialsPoint, 2017)

Ciclo de vida.

Es similar al de una página JSP: el cliente hace una petición HTTP y el servidor responde con la página en HTML. Sin embargo, debido a las características que ofrece JSF, el ciclo de vida incluye algunos pasos más.

El proceso de una petición estándar incluye seis fases, como se muestra en la siguiente figura, representadas por los rectángulos blancos:

Figura 7. Ciclo de vida de JSF



Fuente: capítulo 3. Java Server Faces. [En línea]. <http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/viveros_s_ca/capitulo3.pdf> - [citado el 22 de febrero de 2017]

Los rectángulos grises etiquetados con la leyenda "Process Events" representan la ejecución de cualquier evento producido durante el ciclo de vida.

El funcionamiento de cada etapa se describe brevemente a continuación.

1. Restore View: también llamada Reconstitute Component Tree, es la primera etapa que se lleva a cabo, e inicia cuando se hace una petición. Su objetivo es la creación de un árbol con todos los componentes de la página en cuestión.

2. Apply Request Values: cada uno de los componentes del árbol creado en la fase anterior obtiene el valor que le corresponde de la petición realizada y lo almacena.

3. Process Validations: después de almacenar los valores de cada componente, estos son validados según las reglas que se hayan declarado.

4. Update Model Values: durante esta fase los valores locales de los componentes son utilizados para actualizar los beans que están ligados a dichos componentes. Esta fase se alcanzará únicamente si todas las validaciones de la etapa anterior fueron exitosas.

5. Invoke Application: se ejecuta la acción u operación correspondiente al evento inicial que dio comienzo a todo el proceso.

6. Render Response: la respuesta se renderiza y se regresa al cliente.

Dependiendo del éxito o fracaso de las tareas en cada una de las fases del ciclo de vida, el flujo normal descrito puede cambiar hacia caminos alternos según sea el caso (JavaServer Faces, 2010).

2.5.1.4 Spring MVC

Historia.

Spring es un Framework de aplicación desarrollado por la compañía Interface 21, para aplicaciones escritas en el lenguaje de programación Java. Fue creado gracias a la colaboración de grandes programadores, entre ellos se encuentran como principales partícipes y líderes de este proyecto Rod Johnson y Jürgen Höller. Estos dos desarrolladores, además de otros colaboradores que juntando toda su experiencia en el desarrollo de aplicaciones J2EE (Java 2 Enterprise Editions), incluyendo EJB (Enterprise JavaBeans), Servlets y JSP (Java Server Pages), lograron combinar dichas herramientas y otras más en un sólo paquete, para brindar una estructura más sólida y un mejor soporte para este tipo de aplicaciones.

Además se considera a Spring un Framework lightweight, es decir liviano o ligero, ya que no es una aplicación que requiera de muchos recursos para su ejecución, además el framework completo puede ser distribuido en un archivo .jar de alrededor de 1 MB, lo cual representa muy poco espacio, y para la cantidad de servicios que ofrece es relativamente insignificante su tamaño.

Spring no intenta "reinventar la rueda" sino integrar las diferentes tecnologías existentes, en un sólo framework para el desarrollo más sencillo y eficaz de aplicaciones J2EE portables entre servidores de aplicación.

Otro de los principales enfoques de Spring y por el cual está ganando gran popularidad es que simplifica el desarrollo de aplicaciones J2EE, al intentar evitar el uso de EJB, ya que como menciona Craig Walls en su libro Spring in Action, "En su estado actual, EJB es complicado. Es complicado porque EJB fue creado para resolver cosas complicadas, como objetos distribuidos y transacciones remotas." Y muchas veces aunque el proyecto no es lo suficientemente complejo, se utiliza EJB, contenedores de alto peso y otras herramientas que soportan un grado mayor de complejidad, como una solución a un proyecto. "Con Spring, la complejidad de tu aplicación es proporcional a la complejidad del problema que se está resolviendo." Esto sin embargo no le quita crédito a EJB, ya que también ofrece a los desarrolladores servicios valiosos y útiles para resolver ciertas tareas, la diferencia radica en que Spring intenta brindar los mismos servicios pero simplificando el modelo de programación (UDLAP, 2013).

Spring fue creado basado en los siguientes principios:

1. El buen diseño es más importante que la tecnología subyacente.

2. Los JavaBeans ligados de una manera más libre entre interfaces es un buen modelo.
3. El código debe ser fácil de probar.

Versiones.

La primera versión fue escrita por Rod Johnson, quien lanzó el marco con la publicación de su libro "Expert One-on-One J2EE Diseño y Desarrollo en octubre de 2002.

El marco fue lanzado por primera vez bajo la licencia Apache 2.0 en junio de 2003.

El primer hito Lanzamiento, 1.0, fue lanzado en marzo de 2004, con otros lanzamientos del hito en septiembre de 2004 y marzo de 2005.

El marco 1.2.6 de la primavera ganó un premio de la productividad de la sacudida y un premio de la innovación de JAX en 2006.

Spring 2.0 fue lanzado en octubre de 2006, primavera de 2.5 en noviembre de 2007, primavera de 3.0 en diciembre de 2009, primavera de 3.1 en diciembre de 2011 y primavera de 3.2.5 en noviembre de 2013.

Spring Framework 4.0 fue lanzado en diciembre de 2013. Las mejoras notables en Spring 4.0 incluyen soporte para Java SE 8, Groovy 2, algunos aspectos de Java EE7 y WebSocket.

Spring Framework 4.2.0 fue lanzado el 31 de julio de 2015 y fue inmediatamente actualizado a la versión 4.2.1, que fue lanzado el 01 de septiembre de 2015. Es

"compatible con Java 6, 7 y 8, con un enfoque en refinamientos de núcleo y capacidades web modernas".

Spring Framework 4.3 ha sido lanzado el 10 de junio de 2016. La versión 4.3.0.RC1 está disponible. Será "la última generación dentro de los requisitos generales del sistema Spring 4 (Java 6+, Servlet 2.5+), preparándose para una prolongada vida de soporte 4.3.x hasta 2019" (Faraoni, 2014)

Características.

Separación clara de roles. Cada objeto-controlador, validador, objeto de comando, objeto de formulario, objeto de modelo, DispatcherServlet, mapeo de manejador, visualizador de resolución, etc., pueden ser satisfechos por un objeto especializado.

Configuración potente y sencilla de las clases de Framework y de aplicación como JavaBeans. Esta capacidad de configuración incluye una fácil referencia a través de contextos, como desde controladores web a objetos de negocio y validadores.

Adaptabilidad, no intrusividad y flexibilidad. Define cualquier firma de método de controlador que necesite, posiblemente utilizando una de las anotaciones de parámetros (@RequestParam, @RequestHeader, @PathVariable y más) para un escenario dado.

Código de negocio reutilizable, no hay necesidad de duplicación. Utiliza objetos empresariales existentes como objetos de comando o formulario en lugar de reflejarlos para extender una clase base de marco en particular.

Vinculación y validación personalizables. Escribe desajustes como errores de validación a nivel de aplicación que mantienen el valor ofensivo, la fecha localizada y la vinculación de números, y así sucesivamente, en lugar de los objetos de formulario de sólo cadena con análisis manual y conversión a objetos empresariales.

Mapeo del manejador personalizable y resolución de la vista. Las estrategias de asignación y resolución de vistas del manipulador van desde la simple configuración basada en URL hasta las sofisticadas estrategias de resolución diseñadas específicamente. Spring es más flexible que los Frameworks web MVC que requieren una técnica particular.

Transferencia flexible del modelo. Transferencia de modelo con un nombre / valor. El mapa admite una fácil integración con cualquier tecnología de vista.

Configuración local personalizable. Zona horaria y resolución de temas, compatibilidad con JSP con o sin biblioteca de etiquetas de Spring, soporte para JSTL, soporte para Velocity sin necesidad de puentes adicionales, etc.

Una biblioteca de etiquetas JSP. Sencilla pero potente conocida como la biblioteca de etiquetas de Spring que proporciona soporte para funciones como vinculación de datos y temas. Las etiquetas personalizadas permiten la máxima flexibilidad en términos de código de marcado.

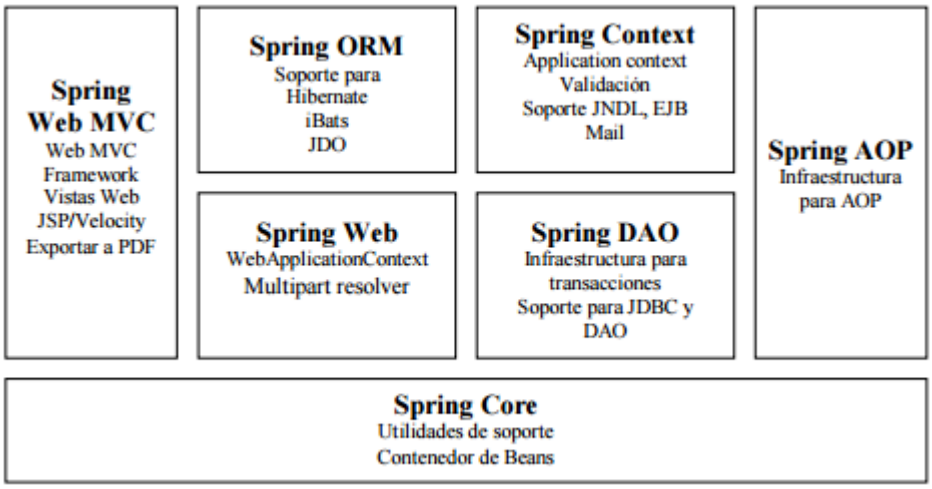
Una biblioteca de etiquetas de formulario JSP. Introducida en Spring 2.0, que hace que escribir formularios en páginas JSP sea mucho más fácil.

Beans cuyos ciclos de vida tienen alcance para la solicitud HTTP actual o sesión HTTP. Esto no es una característica específica de Spring MVC en sí, sino del contenedor WebApplicationContext que Spring MVC utiliza.(Spring, 2012)

Arquitectura.

Spring es un Framework modular que cuenta con una arquitectura divida en siete capas o módulos, como se muestra en la Figura 2.2-4, lo cual permite tomar y ocupar únicamente las partes que interesen para el proyecto y juntarlas con gran libertad.

Figura 8. Arquitectura Spring Framework



Fuente: capítulo 3. Spring un framework de aplicación. [En línea]. <http://catarina.udlap.mx/u_dl_a /tales/documentos/lis/viveros_s_ca/capitulo3.pdf> - [citado el 22 de febrero de 2017]

Spring Core. Esta parte es la que provee la funcionalidad esencial del Framework, está compuesta por el BeanFactory, el cual utiliza el patrón de Inversión de Control (Inversion of Control) y configura los objetos a través de Inyección de Dependencia (Dependency Injection). El núcleo de Spring es el paquete org.springframework.beans el cual está diseñado para trabajar con JavaBeans.

Bean Factory. Es uno de los componentes principales del núcleo de Spring. Es una implementación del patrón Factory, pero a diferencia de las demás implementaciones de este patrón, que muchas veces sólo producen un tipo de objeto, BeanFactory es de propósito general, ya que puede crear muchos tipos diferentes de Beans. Los Beans pueden ser llamados por nombre y se encargan de manejar las relaciones entre objetos.

Todas las Bean Factories implementan la interfaz org.springframework.beans.factory.BeanFactory, con instancias que pueden ser accesadas a través de esta interfaz. Además también soportan objetos de dos modos diferentes:

- **Singleton:** Existe únicamente una instancia compartida de un objeto con un nombre particular, que puede ser regresado o llamado cada vez que se necesite. Este es el método más común y el más usado. Este modo está basado en el patrón de diseño que lleva el mismo nombre.
- **Prototype:** también conocido como non-singleton, en este método cada vez que se realiza un regreso o una llamada, se crea un nuevo objeto independiente.

Inversion of Control. "Inversion of Control se encuentra en el corazón de Spring" BeanFactory utiliza el patrón de Inversión de Control o como se le conoce IoC, que es una de las funcionalidades más importantes de Spring. Esta parte se encarga de separar del código de la aplicación que se está desarrollando, los aspectos de configuración y las especificaciones de dependencia del framework. Todo esto configurando los objetos a través de Inyección de Dependencia o Dependency Injection, que se explicará más adelante.

Una forma sencilla de explicar el concepto de IoC es el "principio Hollywood": "No me llames, yo te llamaré a ti". Traduciendo este principio a términos de este trabajo, en lugar de que el código de la aplicación llame a una clase de una librería, un Framework que utiliza IoC llama al código. Es por esto que se le llama "Inversión", ya que invierte la acción de llamada a alguna librería externa.

Dependency Injection. Es una forma de Inversión de Control, que está basada en constructores de Java, en vez de usar interfaces específicas del Framework. Con este principio en lugar de que el código de la aplicación utilicé el API del Framework para resolver las dependencias como: parámetros de configuración y objetos colaborativos, las clases de la aplicación exponen o muestran sus dependencias a través de métodos o constructores que el Framework puede llamar con el valor apropiado en tiempo de ejecución, basado en la configuración.

Todo esto se puede ver de una forma de push y pop, el contenedor hace un push de las dependencias para ponerlas dentro de los objetos de la aplicación, esto ocurre en tiempo de ejecución. La forma contraria es tipo pull, en donde los objetos de la aplicación jalan las dependencias del ambiente. Además los objetos de la Inyección de Dependencia nunca cargan las propiedades ni la configuración, el Framework es totalmente responsable de leer la configuración.

Spring soporta varios tipos de Inyección de Dependencia, pero en si estos son los dos más utilizados:

- Setter Injection: en este tipo la Inyección de Dependencia es aplicada por medio de métodos JavaBeans setters, que a la vez tiene un getter respectivo.

- Constructor Injection: esta Inyección es a través de los argumentos del constructor.

Spring Context. "El módulo BeanFactory del núcleo de Spring es lo que lo hace un contenedor, y el módulo de contexto es lo que hace un framework".

En sí Spring Context es un archivo de configuración que provee de información contextual al framework general. Además provee servicios enterprise como JNDI, EJB, e-mail, validación y funcionalidad de agenda.

Application Context. ApplicationContext es una subinterfaz de BeanFactory, ya que org.springframework.context.ApplicationContext es una subclase de BeanFactory. En si todo lo que puede realizar una BeanFactory también lo puede realizar ApplicationContext. En sí agrega información de la aplicación que puede ser utilizada por todos los componentes.

Además brinda las siguientes funcionalidades extra:

- Localización y reconocimiento automático de las definiciones de los Beans
- Cargar múltiples contextos
- Contextos de herencia
- Búsqueda de mensajes para encontrar su origen.
- Acceso a recursos
- Propagación de eventos, para permitir que los objetos de la aplicación puedan publicar y opcionalmente registrarse para ser notificados de los eventos.
- Agrega soporte para internacionalización (i18n)

En algunos casos es mejor utilizar ApplicationContext ya que obtienes más funciones a un costo muy bajo, en cuanto a recursos se refiere.

Spring AOP. Aspect-oriented programming, o AOP, es una técnica que permite a los programadores modularizar ya sea las preocupaciones crosscutting, o el comportamiento que corta a través de las divisiones de responsabilidad, como logging, y manejo de transacciones. El núcleo de construcción es el aspect, que encapsula comportamiento que afectan a diferentes clases, en módulos que pueden ser reutilizados.

AOP se puede utilizar para:

1. Persistencia
2. Manejo de transacciones
3. Seguridad
4. Logging
5. Debugging

AOP es un enfoque diferente y un poco más complicado de acostumbrarse en comparación con OOP (Object Oriented Programming). Rob Johnson prefiere referirse a AOP como un complemento en lugar de como un rival o un conflicto.

Spring AOP es portable entre servidores de aplicación, funciona tanto en servidores web como en contenedores EJB.

Spring AOP soporta las siguientes funcionalidades:

- Intercepción: se puede insertar comportamiento personalizado antes o después de invocar a un método en cualquier clase o interfaz.
- Introducción: Especificando que un advice (acción tomada en un punto particular durante la ejecución de un programa) debe causar que un objeto implemente interfaces adicionales.
- Pointcuts dinámicos y estáticos: para especificar los puntos en la ejecución del programa donde debe de haber intercepción.

Spring ORM. En lugar de que Spring proponga su propio módulo ORM (Object-Relational Mapping), para los usuarios que no se sientan confiados en utilizar simplemente JDBC, propone un módulo que soporta los Frameworks ORM más populares del mercado, entre ellos.

- Hibernate (2.1 y 3.0): es una herramienta de mapeo O/R open source muy popular, que utiliza su propio lenguaje de query llamada HQL.
- iBATIS SQL Maps (1.3 y 2.0). una solución sencilla pero poderosa para hacer externas las declaraciones de SQL en archivos XML.
- Apache OJB (1.0): plataforma de mapeo O/R con múltiples APIs para clientes.
- Entre otros como JDO (1.0 y 2.0) y Oracle TopLink.

Spring DAO. El patrón DAO (Data Access Object) es uno de los patrones más importantes y usados en aplicaciones J2EE, y la arquitectura de acceso a los datos de Spring provee un buen soporte para este patrón [Johnson, 2005].

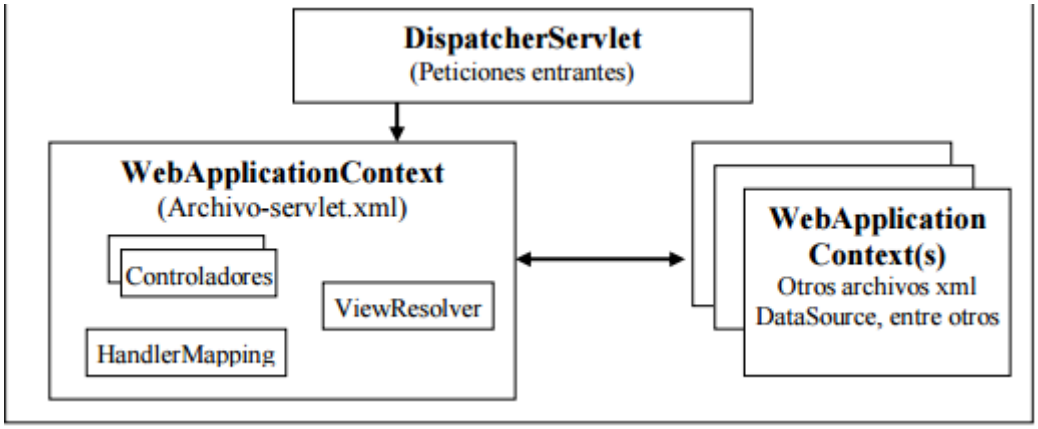
Spring web. El módulo web de Spring se encuentra en la parte superior del módulo de contexto, y provee el contexto para las aplicaciones web. Este módulo también provee el soporte necesario para la integración con el framework Struts de Yakarta.

Este módulo también se encarga de diversas operaciones web como por ejemplo: las peticiones multi-parte que puedan ocurrir al realizar cargas de archivos y la relación de los parámetros de las peticiones con los objetos correspondientes (domain objetcs o business objects).

Spring Web MVC. Spring brinda un MVC (Model View Controller) para web bastante flexible y altamente configurable, pero esta flexibilidad no le quita sencillez, ya que se pueden desarrollar aplicaciones sencillas sin tener que configurar muchas opciones.

Para esto se puede utilizar muchas tecnologías ya que Spring brinda soporte para JSP, Struts, Velocity, entre otros.(UDLAP, 2013)

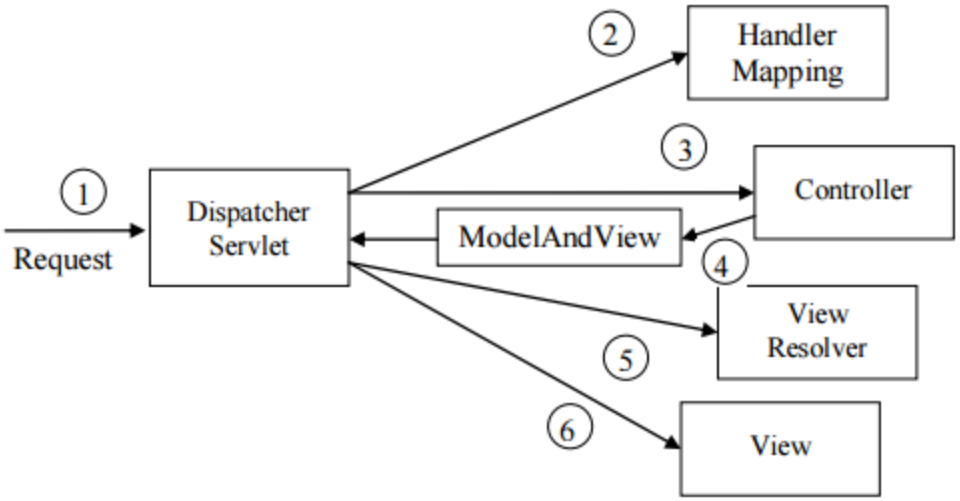
Figura 9. Arquitectura básica de Spring web MVC.



Fuente: capítulo 3. Spring un Framework de aplicación. [En línea]. <http://catarina.udlap.mx/u_dl_a /tales/documentos/lis/viveros_s_ca/capitulo3.pdf> - [citado el 22 de febrero de 2017]

Ciclo de vida.

Figura 10. Ciclo de vida de un request en Spring MVC



Fuente: capítulo 3. Spring un framework de aplicación. [En línea]. <http://catarina.udlap.mx/u_dl_a /tales/documentos/lis/viveros_s_ca/capitulo3.pdf> - [citado el 22 de febrero de 2017]

1. El navegador manda un request y lo recibe un DispatcherServlet.
2. Se debe escoger que Controller manejará el request, para esto el HandlerMapping mapea los diferentes patrones de URL hacia los controladores, y se le regresa al DispatcherServlet el Controller elegido.
3. El Controller elegido toma el request y ejecuta la tarea.
4. El Controller regresa un ModelAndView al DispatcherServlet.
5. Si el ModelAndView contiene un nombre lógico de un View se tiene que utilizar un ViewResolver para buscar ese objeto View que representará el request modificado.
6. Finalmente el DispatcherServlet despacha el request al View.

Spring cuenta con una gran cantidad de controladores de los cuales se puede elegir dependiendo de la tarea, entre los más populares se encuentra: Controller y AbstractController para tareas sencillas; el SimpleFormController ayuda a controlar

formularios y el envío de los mismos, MultiActionController ayuda a tener varios métodos dentro un solo controlador a través del cual se podrán mapear las diferentes peticiones a cada uno de los métodos correspondientes.(UDLAP, 2013).

2.5.1.5 Vaadin

Historia.

El Framework Vaadin no fue escrito de un día para otro. Después de trabajar con interfaces web de usuario desde el comienzo de la web, un grupo de desarrolladores se reunieron en el año 2000 para formar IT Mill. El equipo tenía el deseo de desarrollar un nuevo paradigma de programación que soportara la creación de interfaces de usuario reales para aplicaciones reales usando un lenguaje de programación verdadero.

La librería (biblioteca) fue llamada originalmente Millstone Library. La primera versión fue usada en una aplicación de producción grande que IT Mill diseñó e implementó para una compañía farmacéutica internacional. IT Mill creó la aplicación allá en el año 2001 y todavía está en uso.

Desde entonces, la compañía ha desarrollado docenas de aplicaciones de negocio de gran calibre con la librería y ha mostrado su habilidad para resolver problemas difíciles fácilmente.

La siguiente generación de la librería, IT Mill Toolkit Release 4, fue publicada en el año 2006. Se introdujo un nuevo motor de presentación completamente nuevo basado en AJAX. Esto permitió el desarrollo de aplicaciones AJAX sin necesidad de preocuparse de las comunicaciones entre el cliente y el servidor.22

Versiones.

Versión 5. Entra en el código abierto IT Mill Toolkit 5, publicado inicialmente a finales del año 2007, dio un paso importante en la dirección de AJAX. La renderización del lado del cliente de la interfaz de usuario fue totalmente reescrita usando GWT, Google Web Toolkit.

IT Mill Toolkit 5 presentó muchas mejoras significativas tanto en el API del lado del servidor como en la funcionalidad. La reescritura del motor del lado cliente con GWT permitió el uso de Java tanto en el lado del cliente como en el del servidor. La transición de JavaScript a GWT hizo el desarrollo y la integración de componentes personalizados y la personalización de los componentes existentes mucho más fácil que antes, y también permitió la integración sencilla con componentes GWT ya existentes. La adopción de GWT en el lado del cliente, no produjo por sí misma, ningún cambio en el API del lado del servidor, porque GWT es una tecnología del navegador que está totalmente oculta tras el API. También los temas fueron totalmente revisados en IT Mill Toolkit 5.

La versión 5 fue publicada bajo la licencia Apache License 2, una licencia de código abierto sin restricciones, para impulsar la rápida expansión de la base de usuarios y para hacer posible la creación de una comunidad de desarrolladores.

El nacimiento de Vaadin versión 6. IT Mill Toolkit fue renombrado como Vaadin Framework, o Vaadin de forma corta, en primavera de 2009. Después la compañía, IT Mill, también fue renombrada como Vaadin Ltd. Vaadin quiere decir hembra adulta de reno semidomesticada de la montaña en finlandés.

Con Vaadin 6, el número de desarrolladores usando el framework creció exponencialmente. Junto con la nueva versión, se publicó el complemento (plugin) de Vaadin para Eclipse, que ayuda a la creación de proyectos Vaadin. La introducción del directorio de Vaadin a principios del 2010 dio un impulso adicional, ya que el número de componentes disponibles se multiplicó de la noche a la mañana. Muchos de los componentes inicialmente experimentales han madurado desde entonces y ahora son usados por miles de desarrolladores. En el año 2013, se está viendo un inmenso crecimiento en el ecosistema alrededor de Vaadin. El tamaño de la comunidad de usuarios, al menos si se mide por la actividad del foro, ha sobrepasado a los marcos del lado de servidor competidores e incluso a GWT.

La revisión más importante con Vaadin 7. La versión 7 de Vaadin es la revisión más importante que cambia el API de Vaadin mucho más que lo hizo la versión 6 de Vaadin. Esta más orientado a la web que lo fue Vaadin 6. Se Está haciendo todo lo que está a la mano para ayudar a Vaadin a alzarse alto en el universo web. Parte de este trabajo es fácil y casi rutina, como el arreglar defectos e implementar nuevas funcionalidades, pero ir más alto también requiere permanecer más firme. Ese fue uno de los objetivos de Vaadin 7, rediseñar el producto de forma que la nueva arquitectura permita a Vaadin, superar algunos desafíos que venían de lejos. Muchos cambios requirieron romper la compatibilidad del API con Vaadin 6, especialmente en el lado del cliente, pero se han hecho con la fuerte voluntad de evitar acarrear una carga legada innecesaria en el futuro. Vaadin 7 incluye una capa de compatibilidad para hacer más fácil la adopción de Vaadin 7 para las aplicaciones existentes.

La inclusión de Google Web Toolkit en Vaadin 7 es un desarrollo significativo, pues significa que ahora también se suministra soporte para GWT también. Cuando Google abrió el desarrollo de GWT en verano del 2012, Vaadin (la compañía) se unió al nuevo comité directivo de GWT. Como miembro del comité, Vaadin puede trabajar por el éxito de GWT como fundamento de la comunidad de desarrollo web Java.(Vaadin, 2011)

Características.

Componente de Framework integrados

1. Un gran conjunto de componentes de interfaz de usuario, controles y widgets
2. Widgets ricos e interactivos con carga perezosa incorporada
3. Soporte para eventos de toque móvil
4. Soporte de arrastrar y soltar
5. Crear diseños en Java o en HTML, o ambos
6. Soporta patrones de interfaz de usuario comunes como MVC (model-view-controller) o MVP (model-view-presenter)
7. Crear nuevos componentes con composición y herencia
8. Más de 650 componentes complementarios disponibles en vaadin.com/directory

Compatibilidad Web

9. No se necesitan complementos de navegador
10. Soporta todos los principales navegadores web
11. Soporte de ventanas y pestañas del navegador
12. Soporte de botón de retroceso

13. Soporte de vinculación profunda
14. Parámetros de URL y manejo de fragmentos
15. Soporte de API de historial de HTML5
16. Insertar en cualquier página web
17. Soporte de video y audio HTML5
18. Soporte de importación de HTML5 HTML
19. Servidor incorporado push (utilizando WebSockets, retroceso automático a sondeo (largo) si no es soportado por el navegador / servidor)

Look and Feel Personalizable

20. Potente estilo de componentes basado en CSS y SASS
21. Temas y estilos incorporados atractivos
22. Crear temas de aplicaciones personalizadas sin cambiar el código Java

Desarrollo Java web

23. Sólo en Java: desarrollo web orientado a objetos de tipo seguro
24. Fácil y potente modelo de programación del lado del servidor
25. Gestión de dependencias basada en Maven
26. Compatible con OSGi
27. Compatible con cualquier otro lenguaje JVM como Groovy

Arquitectura de aplicaciones web segura

28. Administración del estado de la interfaz de usuario del servidor
29. Código de aplicación, validaciones y lógica empresarial en el servidor

30. Validación segura de parámetros y solicitudes

31. Soporte de validación JSR-303 incorporado

32. Protección CSRF y soporte SSL

Arquitectura de componentes extensible

33. Arquitectura basada en componentes extensibles

34. Los widgets personalizados pueden ser construidos con:

- GWT
- JavaScript
- WebComponents

35. Fácil, Jar o Zip basado en el widget personalizado de embalaje

36. Distribución desarrollador-desarrollador de complementos en el directorio de Vaadin

Despliegue

37. Implementación basada en Java EE 6+ y Servlet 3.0+ (JSR-154)

38. Despliegue de portlet (JSR-286)

Herramientas

39. Plugin Eclipse IDE

40. Plugin IntelliJ IDEA

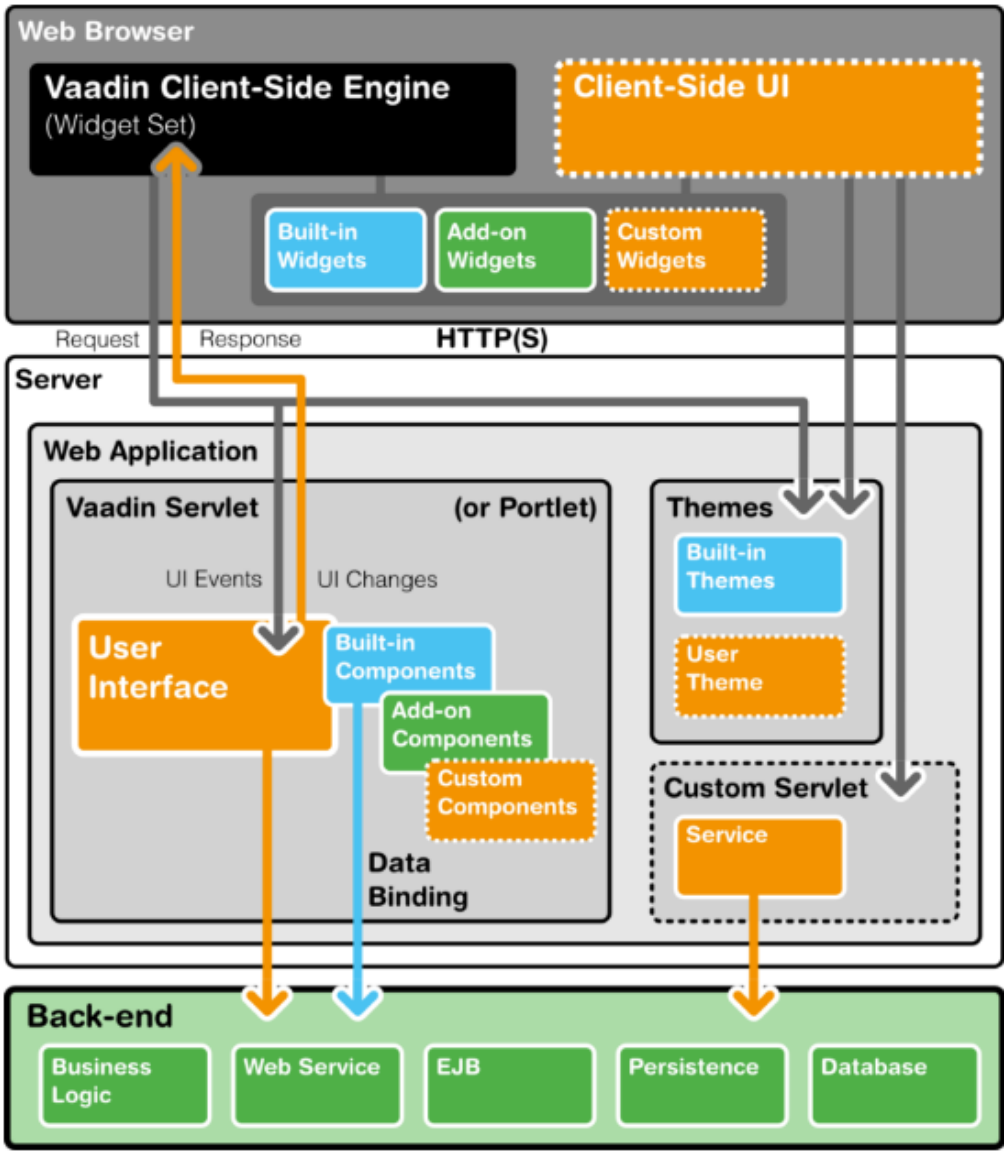
41. Integración con Netbeans IDE

42. Soporta Maven, Gradle e Ivy para el manejo de dependencias

43. Pruebas de interfaz de usuario con Vaadin TestBench y JUnit.(Vaadin, 2017)

Arquitectura.

Figura 11. Arquitectura Framework Vaadin



Fuente: Documentatión for Vaadin. Overview. [en línea]< https://vaadin.com/docs7/-/part7/framework/architecture/architecture-overview.html> - [citado el 22 de febrero de 2017]

La arquitectura de la aplicación del lado del servidor se basa en el framework del lado del servidor (server-side framework) y en un motor del lado del cliente (client-side engine). Este motor se ejecuta en el navegador como código JavaScript, renderizando la interfaz de usuario, y entregando la información de las interacciones de usuario al

servidor. La lógica de interfaz de usuario (UI) de una aplicación se ejecuta como un Servlet Java en el servidor de aplicaciones Java.

Como el motor del lado del cliente se ejecuta como JavaScript en el navegador, no es necesario ningún plugin para el navegador para las aplicaciones hechas con Vaadin. Esto le da una ventaja sobre frameworks basados en Flash, Applets Java o otros complementos (plugins). Vaadin se basa en el soporte de Google Web Toolkit para una amplia gama de navegadores, así que el desarrollador no tiene necesidad de preocuparse por el soporte para navegadores.

Dado que HTML, JavaScript y otras tecnologías del navegador son prácticamente invisibles para la lógica de la aplicación, puedes pensar en el navegador web como una plataforma de cliente ligero. Un cliente ligero muestra la interfaz de usuario y comunica los eventos de usuario al servidor a bajo nivel. La lógica de control de la interfaz de usuario se ejecuta en un servidor web basado en Java, junto con la lógica de negocio. Por contra, una arquitectura corriente cliente-servidor con una aplicación cliente dedicada, incluiría muchas comunicaciones de aplicación especificas entre el cliente y el servidor. En esencia, eliminar la capa interfaz de usuario de la arquitectura de la aplicación hace nuestro muy eficaz a nuestro enfoque.

Detrás del modelo de desarrollo dirigido por el servidor, Vaadin hace el mejor uso de las técnicas AJAX (Asynchronous JavaScript and XML, para más detalles, ver Sección 3.2.3, “AJAX”) que posibilita el crear Aplicaciones Ricas de Internet (RIA) que son tan sensibles e interactivas como las aplicaciones de escritorio.

Además de desarrollar aplicaciones Java del lado del servidor, se puede desarrollar en el lado del cliente creado nuevos componentes en Java, e incluso completas aplicaciones del lado del cliente que se ejecuten exclusivamente en el navegador. El marco Vaadin del lado del cliente incluye Google Web Toolkit (GWT), que proporciona un compilador de Java a JavaScript que se ejecuta en el navegador, así como un marco de interfaz de usuario con todas las funcionalidades. Con este enfoque, Vaadin es completamente Java en ambos lados.

Vaadin utiliza un motor del lado del cliente para renderizar la interfaz de usuario en el navegador de una aplicación del lado del servidor. Todas las comunicaciones entre cliente y servidor están bien ocultas bajo el capó. El de Vaadin está diseñado para ser extensible, e incluso se pueden usar cualquier componente de terceras partes fácilmente, además del repertorio de componentes ofrecido en Vaadin. De hecho, se pueden encontrar cientos de complementos en el Directorio de Vaadin.

El marco Vaadin define una clara separación entre la estructura de la interfaz de usuario y su apariencia, y te permite desarrollarlos separadamente. Nuestro planteamiento para esto son los temas (themes), que controlan la apariencia por CSS y plantillas de páginas HTML (opcionales). Como Vaadin suministra unos temas excelentes por defecto, normalmente no es necesario hacer mucha personalización, pero es posible si se necesita.(Vaadin, 2011)

2.5.2 Conclusiones del capítulo

El contenido de este capítulo deja como conclusión la importancia del uso de Frameworks web MVC open source en Java para el desarrollo de Sistemas software o sitios web empresariales, exponiendo una serie de características que los identifican a cada uno de ellos y todas estas características son evoluciones que han tenido en el tiempo con el fin de facilitar las actividades al desarrollador y así mejorar la experiencia del usuario al navegar teniendo en cuenta que se hace más detención en la lógica de negocio y aspecto visual.

Este capítulo es una base para la interpretación y análisis de la arquitectura de cada uno de los Frameworks web MVC open source mencionados ya que analizando dicha arquitectura se pudo determinar cuáles son las capas de lógica de procesos que contienen y transferencia entre ellas que llevan a cabo para terminar el ciclo de vida de una petición del cliente.

Es fundamental entender que dependiendo del tipo de sistema o sitio web que se pretenda crear hay diferentes opciones en cuanto a Frameworks web MVC open source que se pueden utilizar y depende únicamente del desarrollador escoger cual usar según sus necesidades. También se puede hacer uso de la integración de varios de estos Frameworks aprovechando lo mejor de cada uno en cuanto a características, tiempo de respuesta y facilidad de uso.

3 Comparativo y selección entre JSF y Spring MVC

Este capítulo contiene la especificación del proceso de comparación y selección entre los Frameworks JavaServer Faces y Spring MVC por medio de criterios de evaluación.

Para el propósito de este proyecto el cual es transformar el aplicativo Administrador Vortal en el CIADTI con el fin de hacer más agradable su entorno o vista al cliente pero sin dejar de lado la agilización de tiempos de respuesta, seguridad de la información e integración con la arquitectura actualmente existente, se hizo el siguiente análisis para determinar los dos Frameworks web MVC open source más apropiados según características y arquitectura:

Tabla 5: Descripción de calificación para la selección de dos Frameworks web MVC open source

Calificación	Descripción
1	Sus características y arquitectura no son apropiadas para el propósito del proyecto
2	Sus características son apropiadas pero su arquitectura posee limitaciones considerables para el propósito del proyecto
3	Su arquitectura es apropiada pero sus características poseen limitaciones relevantes para el desarrollo del proyecto
4	Sus características y su arquitectura son apropiadas para el propósito del proyecto

Tabla 6: Asignación de calificaciones para los Frameworks Web MVC open source

	Frameworks web MVC open source			
	Struts	Spring	Vaadin	JSF
Calificación	3	4	3	4

Los Frameworks web MVC open source que resultaron más apropiados para la transformación del administrador Vortal en el CIADTI según sus características y arquitectura son Spring MVC y JavaServer Faces por lo tanto en este capítulo se llevará a cabo un estudio más a fondo de ellos a modo de comparación para seleccionar el mejor de los dos.

3.1 Criterios de evaluación

3.1.1 Funcionalidad.

Uno de los aspectos más importantes que determinan la aceptación de un producto, en este caso software, es la funcionalidad. Este criterio tiene como propósito evaluar si el Framework alcanza los objetivos para los que fue diseñado, es decir, si brinda al usuario la utilidad necesaria para acabar con los problemas que ofrece resolver.

3.1.2 Fiabilidad.

Este criterio está encargado de evaluar la seguridad del Framework, su forma de manejar sesiones de usuarios entre otros elementos necesarios que ayudan al manejo de una navegación segura.

3.1.3 Mantenibilidad

Este criterio busca evidenciar con que tanta documentación cuenta el Framework para facilitar su aprendizaje, además pretende mostrar qué tan frecuentes son las actualizaciones de versiones por parte de su organización desarrolladora.

3.1.4 Índice de interés

Este criterio pretende mostrar qué tanto interés han tenido las personas que utilizan el buscador de Google, por medio de su herramienta Google Trends que permite a través de gráficas observar el comportamiento de la búsqueda especificada en el tiempo.

3.2 Evaluación de JSF y Spring MVC según criterios

3.2.1 JavaServer Faces

3.2.1.1 Funcionalidad

La estructura de una aplicación web en JSF (ver figura 13) es igual a la de una aplicación web tradicional en java, se crea una carpeta con el nombre del proyecto por ejemplo myproyectoJSF y dentro de esta carpeta se crea la subcarpeta Web Pages donde se contienen todas las subcarpetas y archivos necesarios para la vista de la aplicación y el paquete Source Packages donde se contienen todas las clases y sub paquetes necesarios para la lógica de la aplicación. La diferencia que posee una aplicación web JSF radica en que se crea una archivo de configuración llamado web.xml dentro de la subcarpeta WEB-INF (ver figura 14) donde se establecen una serie de parámetros necesarios para el funcionamiento del Framework, como se puede observar en la figura (ver figura 16) es un archivo que contiene especificaciones basadas en etiquetas xml.

Figura 12. Estructura básica aplicación web JSF y tradicional

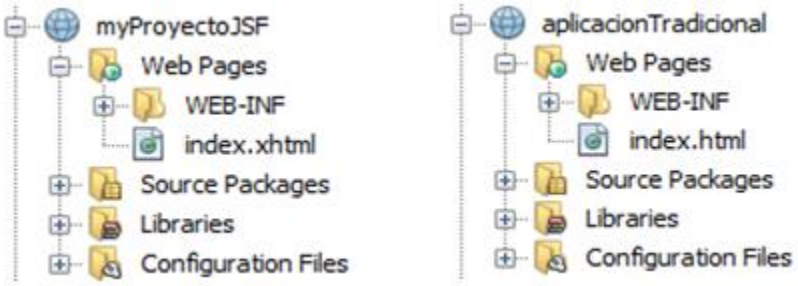
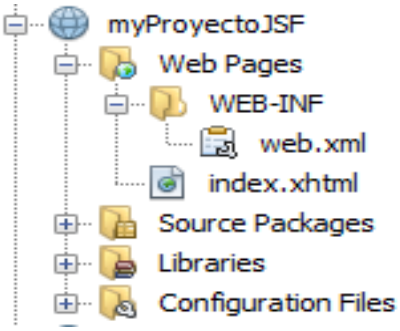
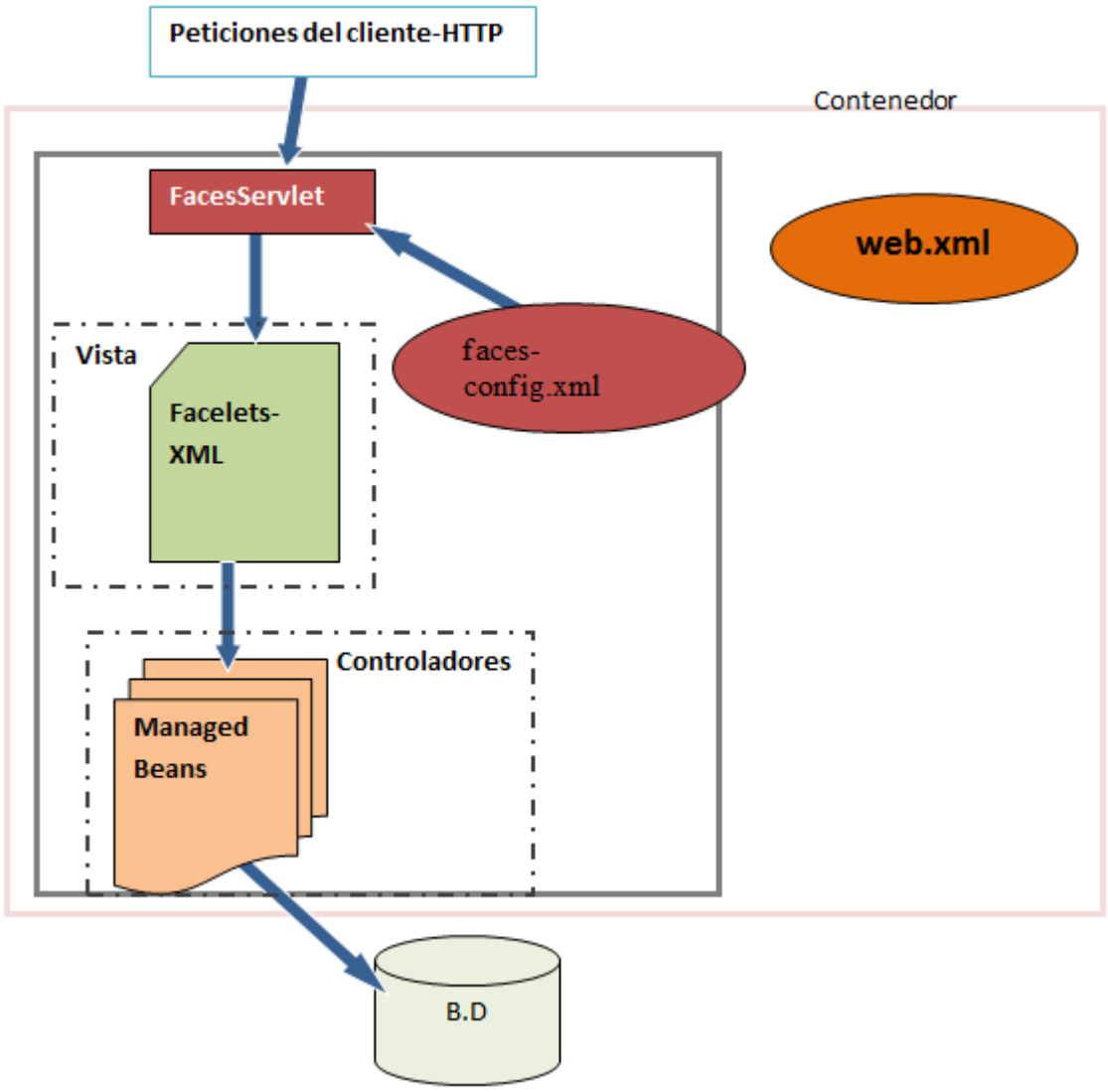


Figura 13 Ubicación archivo de configuración web.xml



La figura 15 muestra un bosquejo del modelo de funcionamiento de JSF y cada uno de los elementos fundamentales que intervienen en su ciclo de vida.

Figura 14. Modelo de funcionamiento JSF



Archivo de configuración web.xml.

Este archivo de configuración se le denomina como un descriptor de despliegue para aplicaciones web basadas en J2EE y que contiene definiciones sobre cómo se debe desplegar la aplicación, estas definiciones son interpretadas por una herramienta de despliegue llamada desplegador que se encarga de dirigir la aplicación con los requisitos especificados.

Figura 15: Configuración del archivo web.xml en JSF

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                             http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>faces/index.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

La etiqueta **web-app** es la raíz del archivo y posee unos atributos especiales que son **versión** donde se especifica la versión de las librerías XML que se están utilizando, el espacio de nombre **xmlns** donde se especifica el URI del esquema que valida el documento web.xml. Cabe resaltar que la versión del esquema debe ser la misma que la versión del XML.

A continuación se hará una explicación de las etiquetas que conforman la configuración básica del archivo web.xml y los valores de sus sub etiquetas.

La etiqueta **context-param** sirve para establecer parámetros por los cuales se va a regir el funcionamiento de la aplicación y contiene además sub etiquetas como **param-**

name que contiene el nombre del parámetro y **param-value** que contiene el valor asignado al parámetro especificado. Por ejemplo:

El primer parámetro (name) de configuración especificado contiene en este caso es el **javax.faces.PROJECT_STAGE**. **Javax** es un prefijo que se utiliza para paquetes de extensiones estándar de java que en este caso incluye el prefijo **faces** que es un paquete de clases de nivel superior del proyecto Java Server Faces API y por último **PROJECT_STAGE** es una clase que sirve para conocer el estado actual de la aplicación desde el momento de su ejecución hasta cuando acabe su ciclo de vida con el llamado al método **Application.getProjectStage()**. Por medio de esta clase JSF permite mostrar mensajes de excepción más específicos de su código de implementación.

El segundo parámetro especificado (value) contiene en este caso el argumento **Development,** el cual indica que la aplicación que se está ejecutando actualmente se encuentra en proceso de desarrollo, pero también puede especificarse este valor con alguno de los siguientes argumentos: **Production** indica que la aplicación se encuentra en etapa de producción, **SystemTest** indica que la aplicación está pasando por pruebas del sistema y por último **UnitTest** indica que la aplicación está siendo sometida a pruebas de unidad.

En la implementación de una aplicación JSF se pueden especificar otros tipos de parámetros para indicarle recursos que la aplicación necesita utilizar por ejemplo si se usa las librerías del Framework de diseño Primefaces y se desea incluir uno de los temas predeterminados de éste, se debe especificar de la siguiente forma:

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

```
    <param-value>nombre del tema</param-value>
</context-param>
```

La etiqueta **servlet** sirve para establecer cuál será la interfaz de programación que se encargará de controlar las solicitudes hechas del lado del cliente y la respuesta que obtendrá (este comportamiento será aclarado con mayor claridad en el punto Configuración del controlador FacesServlet), el primer argumento es **servlet-name** donde se define el nombre para el servlet, el segundo argumento es **servlet-class** que especifica la clase compilada que ejecuta el servlet definido y por último inicializa el servlet asignando como valor 1 para la sub etiqueta **load-on-startup**.

En la etiqueta **servlet-mapping** se le especifica al servlet el patrón de todas las URLs de la aplicación a desarrollar que lo podrán llamar, por defecto JSF establece el patrón /faces/* pero este puede ser modificado a conveniencia.

La etiqueta **sesión-config** permite establecer por medio de la sub etiqueta **sesión-timeout** el tiempo que durará la sesión de un usuario inactiva, es decir la sesión se cerrará automáticamente si el usuario no está utilizando recursos de la aplicación por el tiempo establecido, este tiempo está dado en minutos y por defecto JSF establece 30 como argumento.

Por último la etiqueta **welcome-file-list** contiene una lista de archivos de bienvenida le la aplicación, la sub etiqueta **welcome-file** sirve para especificar el nombre del archivo de bienvenida, por defecto JSF indica que es index.xhtml.

Funcionamiento de la clase controladora FacesServlet.

La etiqueta **servlet** especificada en el archivo web.xml hace el llamado a la clase **FacesServlet,** la cual se encarga de recibir todas las peticiones HTTP e interactuar con la aplicación, es decir este actúa como intermediario controlando todo tipo de peticiones de servicios y asignando recursos dependiendo de las instrucciones definidas en el archivo web.xml, pero además a este controlador se le puede manipular su comportamiento por medio del archivo de configuración **faces-config.xml**, este archivo es opcional puesto que si no se configura, el controlador **FacesServlet** tomará instrucciones de las anotaciones java como por ejemplo **@ManagedBean** que sirve para indicar que una clase java es de tipo Bean administrado por el servidor, **@PostConstruct** que sirve para establecer que el método debe construirse después de ser instanciado por un elemento de la vista, entre otras anotaciones.

Funcionamiento del lenguaje Facelets.

Facelets es el lenguaje que utiliza actualmente Java Server Faces para el manejo de elementos en la vista mediante plantillas de estilos HTML.

Las bibliotecas que se pueden incluir en la vista de un proyecto utilizando el lenguaje facelets son las descritas en la siguiente tabla:

Tabla 7: biblioteca de etiquetas JSF facelets

Biblioteca de etiquetas	**Prefijo**	**URI**	**Componentes**	**Contenido**
Facelets	ui:	http://xmlns.jcp.org/jsf/facelets	ui:composition ui:define ui:component	Etiquetas para

			ui:decorate ui:debug ui:include ui:insert ui:repeat ui:remove ui:param ui:fragment	manejo de plantillas
HTML	h:	http://xmlns.jcp.org/jsf/html	h:head h:body h:button h:column h:commandButton h:commandLink h:dataTable h:form h:graphicImage h:inputFile h:inputHidden h:inputSecret h:inputText h:inputTextarea h:link h:message h:outputText h:doctype h:messages h:outputFormat h:outputLabel h:outputLink h:outputScript h:outputStylesheet h:panelGrid h:panelGroup h:selectBooleanCheckbox h:selectManyCheckbox h:selectManyListbox h:selectManyMenu h:selectManyListbox h:selectOneMenu h:selectOneRadio	Etiquetas que utiliza JSF para manejo de objetos UICompo nent
Core	f:	http://xmlns.jcp.org/jsf/core	f:actionListener f:ajax f:attribute f:attributes	Etiquetas para acciones

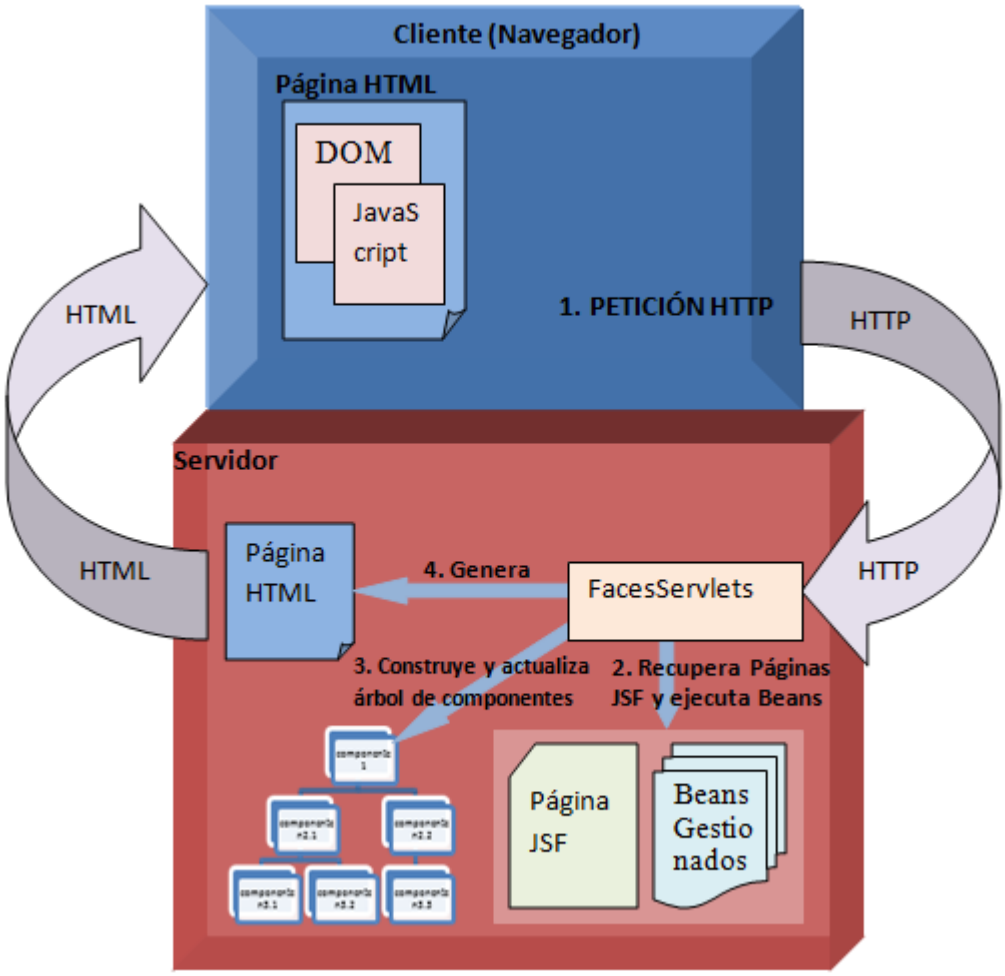
			f:convertDateTime f:convertNumber f:converter f:event f:facet f:loadBundle f:metadata f:param f:passThroughAttribute f:passThroughAttributes f:phaseListener f:selectItem f:selectItems f:setPropertyActionListener f:subview f:validateBean f:validateDoubleRange f:validateLength f:validateLongRange f:validateRegex f:validateRequired f:validator f:valueChangeListener f:verbatim f:view f:viewAction f:viewParam	personalizadas de JSF
JSTL Core	c:	http://xmlns.jcp.org/jsp/jstl/core	c:catch c:choose c:forEach c:if c:otherwise c:set c:when	Etiquetas principales de JSTL
JSF Composite	cc:	http://xmlns.jcp.org/jsf/composite	cc:actionSource cc:attribute cc:clientBehavior cc:editableValueHolder cc:extensión cc:facet cc:implementation cc:insertChildren cc:insertChildren cc:insertFacet cc:interface cc:renderFacet	Etiquetas que contienen solo componentes JSF compuestos

			cc:valueHolder	

PrimeFaces, RichFaces e IceFaces son frameworks desarrollados para Java Server Faces y ofrecen la posibilidad de integrar sus librerías al proyecto y por lo tanto una gran cantidad de etiquetas para el mejoramiento de la interfaz de usuario que pueden interactuar con beans administrados por medio del lenguaje EL (Expression Language).

¿Cómo construye JSF la vista? La primera vez que se genera una página JSF, el navegador realiza una petición de la determinada URL que desea cargar donde se encuentran los componentes JSF que se quieren mostrar, dicha petición es hecha al motor FacesServlets quien se encarga de cargar el árbol de componentes según la solicitud, este comportamiento se describe a continuación:

Figura 16: Construcción de la vista JSF por primera vez



Por ejemplo si en la página JSF se definen elementos como h:commandButton, h:inputText, etc. En el árbol de componentes se construye un objeto de la clase javax.faces.component.html.HtmlCommandButton, javax.faces.component.html.HtmlInputText, etc.

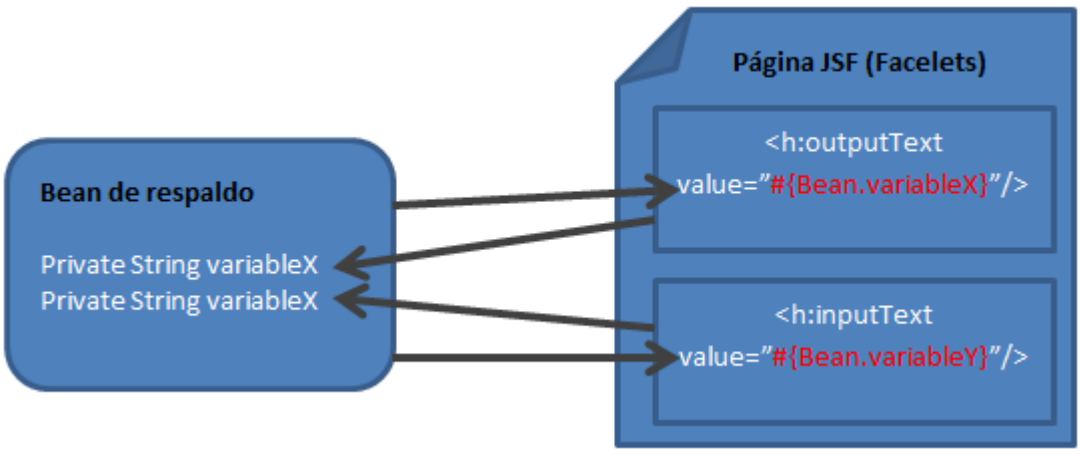
Después de ser construido el árbol de componentes necesarios, se ejecuta en el servidor FacesServlets y este se encarga de rellenar los elementos del árbol con los datos

establecidos en la aplicación y construir la página HTML que se pasa al navegador para mostrar en la vista.

Funcionamiento de los Managed Beans.

Un Bean en java es una clase de respaldo que controla la interacción entre la interfaz de usuario y el modelo, además contiene propiedades y métodos oyentes de eventos que procesan a dichas propiedades. JSF permite asociar los Bean con componentes de formulario en la vista por medio de declaraciones EL y de esta forma a medida que cambia el valor del componente se cambia el estado del Bean y viceversa por la sincronización que hay entre ellos, de tal forma que si se tiene en la página el componente <h:inputText value="#{Bean.variableX}"/> el Bean de respaldo para esa página se construirá al momento de cargar el componente inputText en la vista y su variable variableX tomará el valor ingresado en el campo de texto.

Figura 17: Interacción entre vista y Bean en JSF



3.2.1.2 Fiabilidad

El solo hecho de trabajar con un Framework para aplicaciones web ya implica cierto nivel de seguridad en comparación con aplicaciones creadas manualmente, debido a que estos incluyen ficheros de configuración donde se establecen reglas de navegación y donde se especifican permisos de usuario y roles, así como tiempo de duración de sesión entre otros aspectos.

Para cualquier aplicación web que en su funcionamiento incluya acceso y manipulación de datos de una Base de Datos se requiere tener en cuenta dos aspectos importantes que son la Autenticación y Autorización que hacen referencia a verificación de la autenticidad de los datos del usuario y permiso para acceso a recursos a cada usuario respectivamente.

JSF tiene la ventaja que por ser parte del estándar JEE puede hacer uso de los estándares que hagan parte de este, para el manejo de seguridad en las aplicaciones existe JAAS (Java Authentication and Authorization Service) el cual se encarga de controlar la validación del usuario por medio del logueo y asignar recursos dependiendo de los roles a los cuales tenga permitido acceder el usuario logueado.

Para explicar de manera entendible la forma como JSF maneja la seguridad en las aplicaciones web se dividirá en dos partes que son autenticación y autorización.

Autenticación.

La autenticación es lo que en cualquier aplicación se conoce como el proceso de logueo por medio de una contraseña y un usuario donde el sistema se encarga antes de asignar recursos de verificar la autenticidad de las credenciales en una Base de Datos y después permitir o no el acceso a la aplicación.

En JSF se autentica consultando a la Base de Datos desde Bean administrados por medio de sus anotaciones, por ejemplo si se desea validar si las credenciales de un usuario son auténticas al momento de loguearse, el Bean tendría la siguiente lógica:

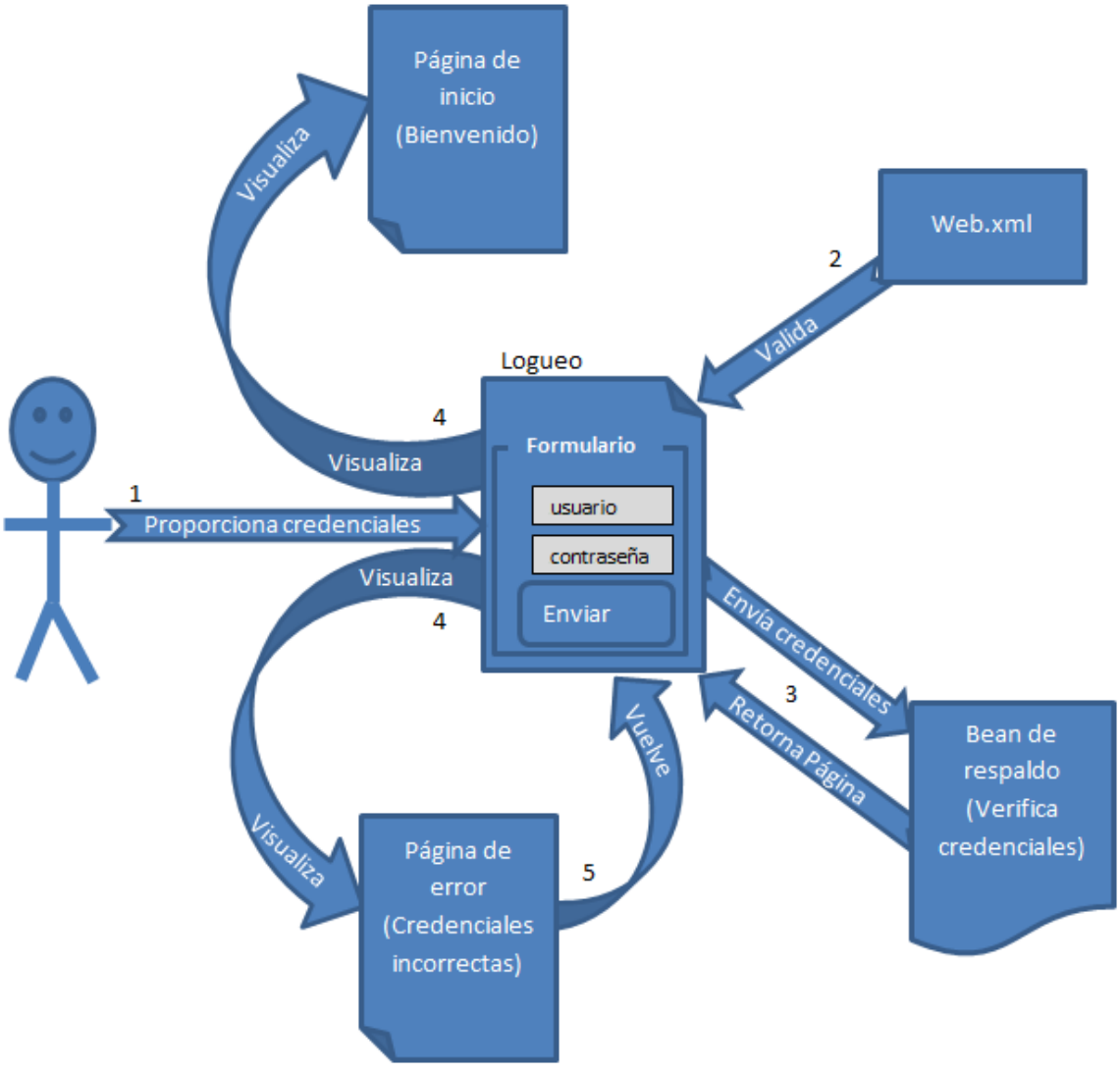
```
@Named(value = "BeanLogueo") //establece el nombre con el cual será accedido el Bean
@SessionScoped //declara el Bean de tipo sesión para que perdure durante la misma
public class Logueo implements Serializable {

   @Inject
   private LogueoDao logueoDao; //inyecta las dependencias de la clase LeogueoDao

public Logueo (){
}

public String autenticar(){
     String mensaje="";
     if(logueoDao.autenticar(usuario)){ //verfica si el usuario existe
          RequestContext.getCurrentInstance()
                                        .getExternalContext()
                                        .getSessionMap()
                                        .put("usuario", usuarioLogueado); // crea el usuario
          mensaje="paginas/inicio/inicio.xhtml"; // envía la ruta de inicio
     }else{
          mensaje="paginas/error.xhtml"; //envía la ruta de error
      }
  return mensaje; //retorna ruta según el caso
}
```

JSF utiliza la clase **FacesContext.getCurrentInstance().getExternalContext(). getSessionMap().put("usuario", usuarioLogueado)** para crear la sesión al usuario logueado, para el ejemplo se especifica un nombre de atributo (“usuario”) y un objeto (“usuarioLogueado”) de tal forma que este usuario podrá ser recuperado y comprobado en cualquier momento mientras dure la sesión por medio de **FacesContext.getCurrentInstance().getExternalContext().getSession(false).getAttribute("usuario").**

En el fichero web.xml se establece quién será la página encargada de llevar a cabo el logueo, por medio de un formulario y cuál será la página de error en caso de no ser auténticas las credenciales proporcionadas.

```
<login-config>
  <auth-method>FORM</auth-method>
   <form-login-config>
     <form-login-page>/paginas/logueo.xhtml</form-login-page>
     <form-error-page>/paginas/errorLogueo.xhtml</form-error-page>
  </form-login-config>
 </login-config>
```

Figura 18: Estructura de logueo en JSF



De acuerdo con el esquema, el flujo del logueo de usuario es: primero el usuario proporciona las credenciales en los campos del formulario en la página de logueo; segundo el servidor por medio de la declaración o regla especificada en el archivo web.xml verifica que coincida la página de logueo y demás parámetros que se hayan establecido; tercero se le envían las credenciales ingresadas por el usuario al Bean de respaldo para que verifique su autenticidad en la Base de Datos y retorne el String de la página a mostrar (inicio o error); y por último la página de logueo solo re-direcciona hacia donde le indique el Bean, en caso de ser hacia la página de error el usuario debe volver a la página de logueo y proporcionar nuevamente las credenciales correctas.

Solo queda saber cómo se controla el cierre de sesión para los usuarios en JSF y para este propósito se debe recuperar la sesión por medio de la clase **HttpSession** ubicada en el paquete **javax.faces.context** y las instrucciones son las siguientes:

FacesContext.getCurrentInstance().getExternalContext().getSessionMap().clear(); Para limpiar el objeto **Map** de la sesión y **FacesContext.getCurrentInstance().getExternalContext().getSession(false).invalidate();** para invalidar la sesión del usuario.

Autorización.

Seguridad Declarativa: Para la confianza durante la navegación de un usuario, JSF proporciona la seguridad declarativa para proteger archivos, directorios y servlet, que consiste en configurar en el archivo web.xml unos parámetros o reglas de navegación mediante declaraciones como **security-role** que sirve para establecer roles en la

aplicación y **security-constraint** que sirve para establecer las reglas o recursos web específicos a los que pueden tener acceso los roles creados.

Por ejemplo:

Si se desea crear dos roles Administrador y Usuario y asignarles diferentes recursos se crearían de la siguiente forma:

Primero se establecen los roles.

```
 <security-role>
      <description>rol de administrador </ description >
      <role-name>Administrador</role-name>
</security-role>
<security-role>
      <description> rol de usuario </description>
      <role-name>Usuario</role-name>
</security-role>
```

Después se le asignan recursos a cada rol.

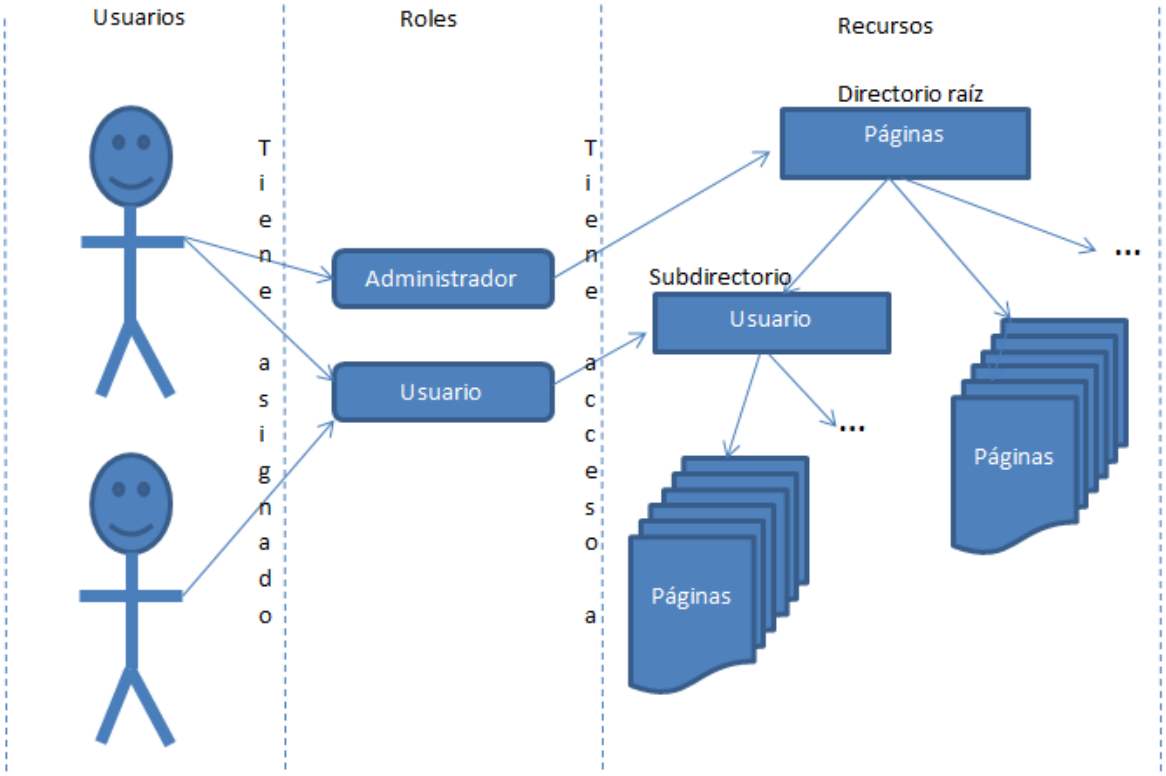
```
<security-constraint>
   <display-name>RecursosAdministrador</display-name>
      <web-resource-collection>
        <web-resource-name>admin</web-resource-name>
        <description/>
        <url-pattern>Paginas/*</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
      </web-resource-collection>
      <auth-constraint>
         <description/>
         <role-name>Administrador</role-name>
      </auth-constraint>
</security-constraint>
<security-constraint>
   <display-name>RecursosUsuario</display-name>
      <web-resource-collection>
        <web-resource-name>usuario</web-resource-name>
        <url-pattern>Paginas/Usuario/*</url-pattern>
```

```
        <http-method>GET</http-method>
        <http-method>POST</http-method>
      </web-resource-collection>
      <auth-constraint>
         <role-name>Usuario</role-name>
      </auth-constraint>
</security-constraint>
```

Como se puede observar lo que se busca establecer en la aplicación por medio del patrón url (url-pattern) es garantizar que el rol Administrador tenga acceso a todas las páginas mientras que el rol Usuario solo tenga acceso a las páginas ubicadas en su directorio Paginas/Usuario ya sea por medio de método POST o GET.

Esquematizando el ejemplo descrito anteriormente se podría visualizar de la siguiente manera:

Figura 19: Esquema manejo de roles con seguridad declarativa en JSF



El uso de reglas en el archivo web.xml garantiza seguridad en cuanto a acceso a las páginas, Sin embargo no garantiza la seguridad de acceso a los Beans de respaldo de la aplicación.

Configuración por anotaciones: Para la extensión de seguridad en cuanto a acceso a recursos ofrecidos por métodos y variables de Beans administrados, JSF implementa las clases personalizadas SecureActionListener y SecureNavigationHandler que se encargan de analizar cada acción del usuario y comprobar la autenticación y autorización durante la navegación. Las clases mencionadas se declaran en el archivo de configuración web.xml de la siguiente forma.

```
<application>
    <action-listener>
        br.com.globalcode.jsf.security.SecureActionListener
    </action-listener>
    <navigation-handler>
        br.com.globalcode.jsf.security.SecureNavigationHandler
    </navigation-handler>
</application>
```

La clase SecureActionListener se encarga de la autenticación, es decir, de interceptar las llamadas que se hacen desde cualquier página a métodos de Bean administrados y comprueba los permisos de dichos métodos.

La clase SecureNavigationHandler se encarga de la autorización, es decir, de reenviar al usuario a la página que solicitó si ya se comprobó que tiene las credenciales necesarias para accederla.

En los métodos de Bean administrados se debe utilizar la anotación @SecurityRoles especificando qué roles los pueden acceder, para ilustrar este proceso se hará uso del mismo ejemplo explicado anteriormente en la seguridad declarativa.

Si se tiene en la vista un formulario con dos botones uno para listar información de usuarios y otro para actualizar un usuario que llaman a los métodos listar y actualizar del Bean respectivamente.

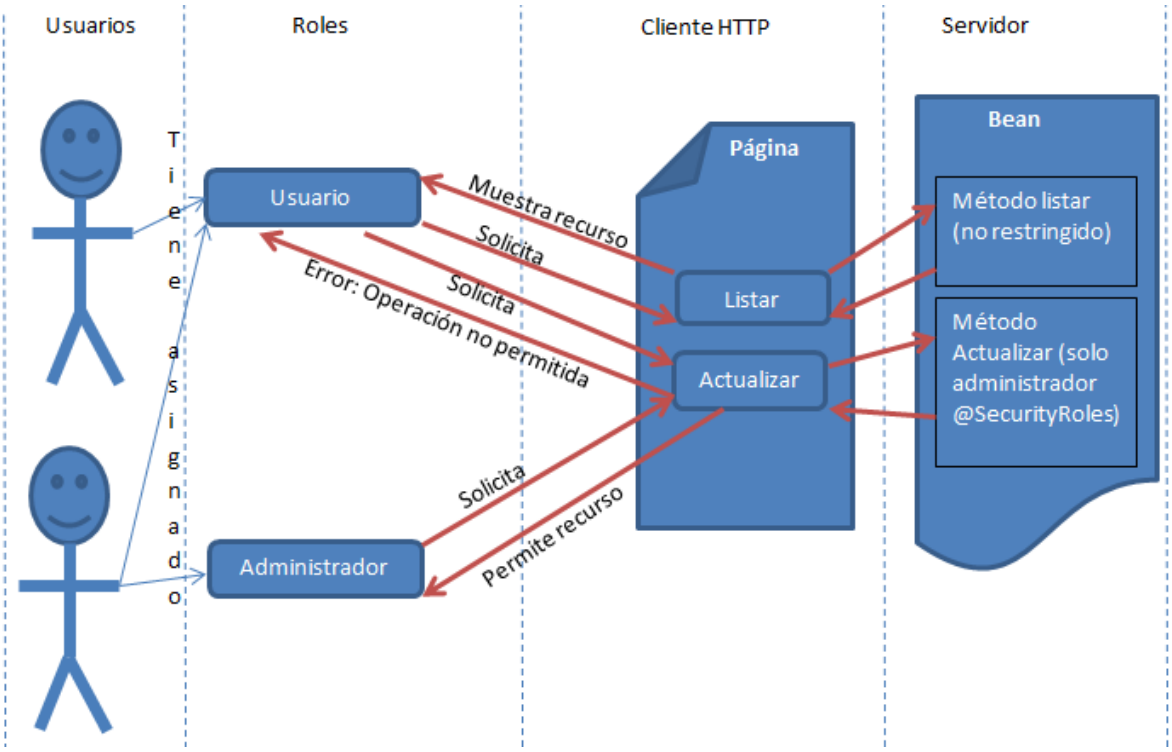
```
<h:form id="seguridadPorAnotaciones">
    <h:commandButton value="Listar" id="btnAccesoTodos"
            action="#{Bean.listar}"/>
    <h:commandButton value="Actualizar"
            id="btnSoloAccesoAdmin"
            action="#{Bean.actualizar}"/>
</h:form>
```

En el Bean administrado el método actualizar debe tener especificado por medio de la anotación @SecurityRoles que solo puede ser accedido por el rol administrador, mientras que el método listar al no tener ninguna restricción puede ser accedido por cualquier rol, de la siguiente forma:

```
public class Bean {

    public List listar() {
          …
      return ListaUsuarios;
    }

    @SecurityRoles("customer-admin-adv, Administrador")
    public String actualizar() {
          …
      return "Usuario actualizado";
}
```

Se puede notar que las restricciones de recursos por anotaciones son más específicas que por el método de seguridad declarativa, ya que dinámicamente se anota qué métodos pueden ser accedidos por qué rol.

Figura 20: Esquema manejo de roles con anotaciones en JSF



En la gráfica se observa como es el funcionamiento de las restricciones de roles, el segundo usuario aunque tiene asignados los dos roles Administrador y Usuario, si se encuentra dentro del sistema con el rol Usuario e intenta accionar el botón actualizar no se le permite realizar la operación debido a que esta funcionalidad solo está permitida para el rol Administrador.

La implementación de seguridad dentro de las aplicaciones realizadas con el Framework JSF tienen particularmente la forma de ser configuradas como se explicó

anteriormente, sin embargo, JSF también permite implementar su seguridad integrando el Framework Spring Security u otros desarrollados para dicho propósito.

3.2.1.3 Mantenibilidad

Para el análisis de mantenibilidad del Framework JavaServer Faces se hará una tabla describiendo las versiones más relevantes a lo largo del tiempo desde que fue lanzado hasta la actualidad y luego una gráfica que muestre la variabilidad de tiempo y esfuerzo que emplean para conservar y mejorar su funcionamiento.

La siguiente tabla describe 5 columnas donde se especifican la versión de JSF, que consiste en la numeración estipulada por el desarrollador; su fecha de lanzamiento, que consiste en el día/mes/año en que fue oficialmente publicada la actualización; el Java Specification Request (JSR), el cual es un proceso formal de Java Community Process (JCP) que consiste en describir la especificaciones y tecnologías propuestas para que el comité ejecutivo (JCP) lo revise antes que sean añadidas a la plataforma Java; descripción, hace referencia a actualizaciones o mejoras incluidas en cuanto a soporte de nuevas tecnologías; y por último la Web oficial del JSR donde se encuentran las descripciones y soportes a los cambios.

Tabla 8: Mantenibilidad de JSF mediante proceso de especificación JSR

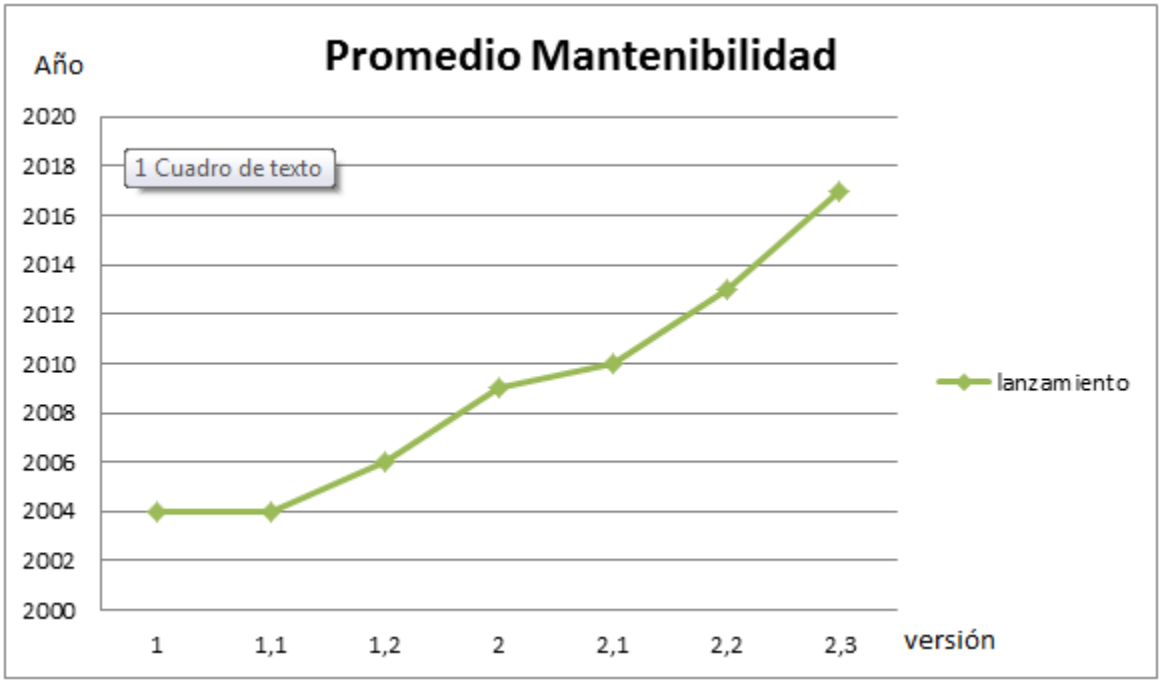
Versión	**JSR**				**Descripción**
	#	Proceso	Inicio	Fin	
		Primera votación de revisión	15-05- 2001	29-05- 2001	

1.0	127	Formación de grupos de expertos	30-05-2001	07-2002	Fue su primer lanzamiento
		Revisión de la comunidad JCP	09-07-2002	12-08-2002	
		Proceso de votación	06-08-2002	12-08-2002	
		Revisión pública	04-03- 2003	18-04-2003	
		Segunda revisión pública	04-06-2003	04-07-2003	
		Revisión final	22-12-2003		
		Votación de aprobación final	17-02-2004	01-03-2004	
		Lanzamiento	11-03-2004		
1.1		Revisión de mantenimiento	15-04-2004	17-05-2004	Incluyó mejoras de errores pero Sin cambios en el renderkit de HTML
		Segundo lanzamiento final	27-05-2004		
1.2	252	Votación de revisión	31-08-2004	13-09-2004	Lanzamiento hecho para corregir errores de funcionamiento en la versión 1.1
		Formación de grupos de expertos	14-09-2004	04-11-2004	
		Revisión del borrador	08-12-2004	07-01-2005	
		Revisión pública	14-04-2005	16-05-2005	
		Votación de revisión pública	10-05-2005	16-05-2005	
		Revisión final	25-08-2005		
		Segunda revisión preliminar	15-02-2006		
		Boletín de aprobación final	18-04-2006	01-05-2006	
		Lanzamiento final	11-05-2006		
		Otros mantenimientos	28-03-2008	25-08-2008	
2.0	314	Votación de revisión	22-05-2007	04-06-2007	Este lanzamiento incluyó mejoras en
		Formación de grupos de expertos	05-06-2007		

		Primera y segunda revisión preliminar	02-06-2008	12-01-2009	cuanto a funcionalidad, rendimiento y empezó a incluir mejoras en la facilidad de uso.
		Votación y revisión pública	26-11-2008	12-01-2009	
		Propuesta final	03-04-2009		
		Votación para aprobación final	12-05-2009	26-05-2009	
		Lanzamiento	01-07-2009		
2.1		Revisión de primer mantenimiento	07-06-2010	16-07-2010	Este lanzamiento no tenía muchos cambios con respecto a la versión 2.0 solo se hicieron algunos mantenimientos de funcionalidad.
		Revisión de segundo mantenimiento	22-09-2010	22-10-2010	
		Liberación de mantenimiento	22-11-2010		
2.2	344	Votación de revisión	01-03-2011	14-03-2011	Este lanzamiento fue muy importante para los desarrolladores ya que introduce soporte a HTML 5, Faces Flow, Stateless views y Resource library contracts.
		Formación de grupos de expertos	15-03-2011	09-06-2011	
		Revisión del borrador	08-11-2011	08-12-2011	
		Votación y revisión pública	14-12-2012	28-01-2013	
		Propuesta final	14-03-2013		
		Votación de aprobación final	02-04-2013	15-04-2013	
		Lanzamiento	21-05-2013		
2.3	372	Votación y revisión	26-08-2014	22-09-2014	Es la versión actual de JSF incluye Facelets, EL, Mojarra entre otros componentes que lo hacen uno de los mejores.
		Formación de grupos de expertos	23-09-2014	28-01-2015	
		Revisión preliminar y pública	21-10-2015	21-02-2017	
		Votación para revisión pública	21-02-2017	06-03-2017	
		Propuesta final	09-03-2017		
		Lanzamiento	07-04-2017		

Como se puede observar en la tabla anterior el trabajo de la comunidad desarrolladora de JSF es constante desde el año 2001 cuando inició el proyecto, la siguiente gráfica describe con mejor claridad el trabajo de mantenimiento del Framework.

Figura 21: Mantenibilidad entre versiones de JSF



3.2.1.4 Índice de interés

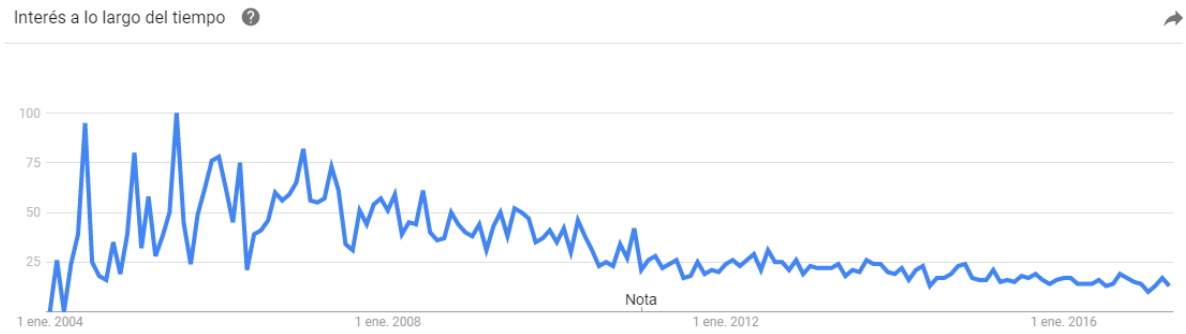
Google es actualmente el buscador más utilizado a nivel de Colombia y la mayoría de países del mundo, además posee diferentes herramientas que permiten facilitar información al usuario, una de estas es Google Trends que muestra una línea de tiempo o histórico de búsqueda de una frase específica, para el caso de estudio se realizó la consulta con tres frases **JavaServer Faces**, **JSF** y **JSF Primefaces** los resultados arrojados son los descritos a continuación.

- ✓ **Búsqueda por la frase JavaServer Faces**

Figura 22: Interés en la búsqueda por la frase JavaServer Faces a nivel mundial Google Trends

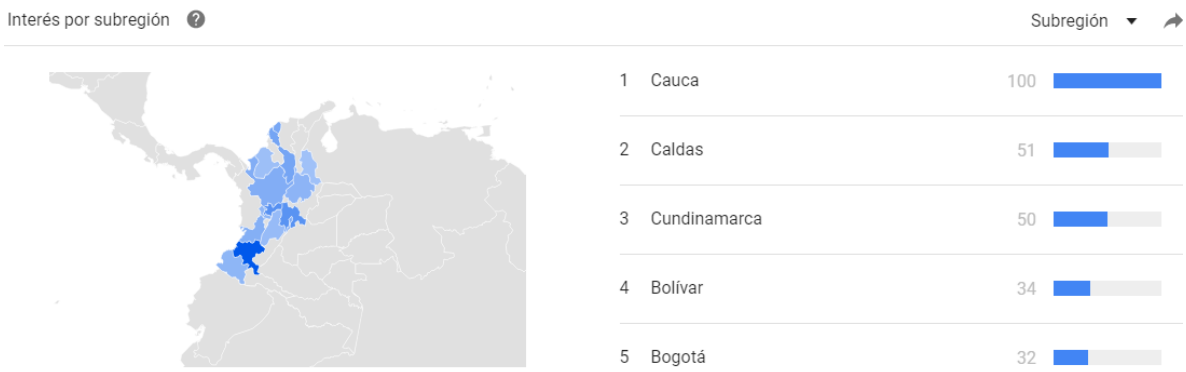
Fuente: Google Trends - <https://trends.google.com/trends/explore?date=all&q=%2Fm%2F026mhl> [citado el 30 de abril de 2017]

Figura 23: Interés en la búsqueda por la frase JavaServer Faces a nivel nacional Google Trends



Fuente: Google Trends - < https://trends.google.com/trends/explore?date=all&geo=CO&q=%2Fm%2F026mhl> [citado el 30 de abril de 2017]

Figura 24: Interés en la búsqueda por la frase JavaServer Faces a nivel regional Google Trends



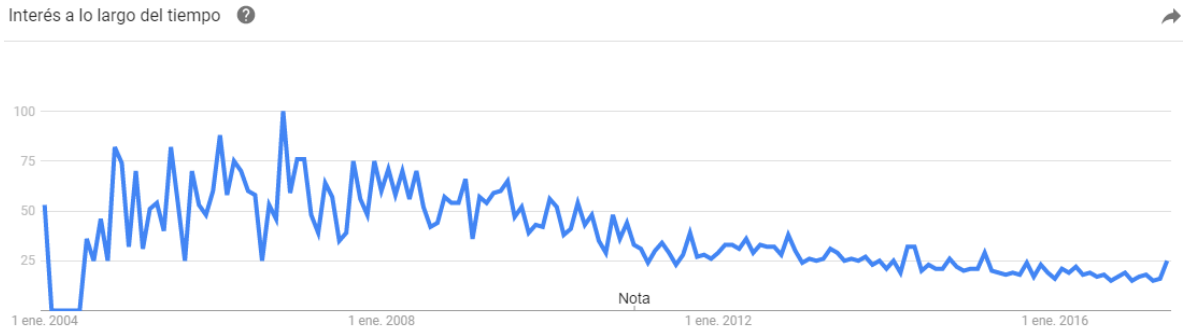
Fuente: Google Trends - <https://trends.google.com/trends/explore?date=all&geo=CO&q=%2Fm%2F026mhl> [citado el 30 de abril de 2017]

✓ **Búsqueda por la palabra JSF**

Figura 25: Interés en la búsqueda por la palabra JSF a nivel mundial Google Trends

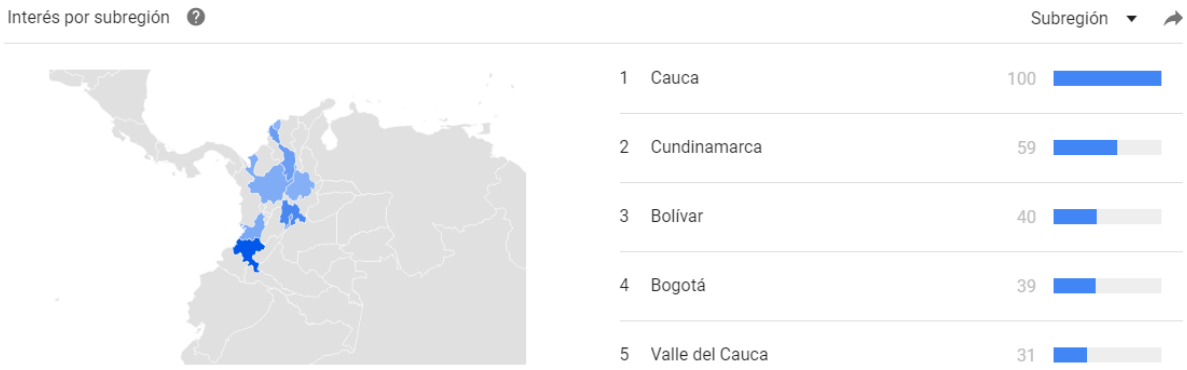
Fuente: Google Trends - <https://trends.google.com/trends/explore?date=all&geo=CO&q=%2Fm%2F026mhl> [citado el 30 de abril de 2017]

Figura 26: Interés en la búsqueda por la palabra JSF a nivel nacional Google Trends



Fuente: Google Trends - <https://trends.google.com/trends/explore?date=all&geo=CO&q=%2Fm%2F026mhl> [citado el 30 de abril de 2017]

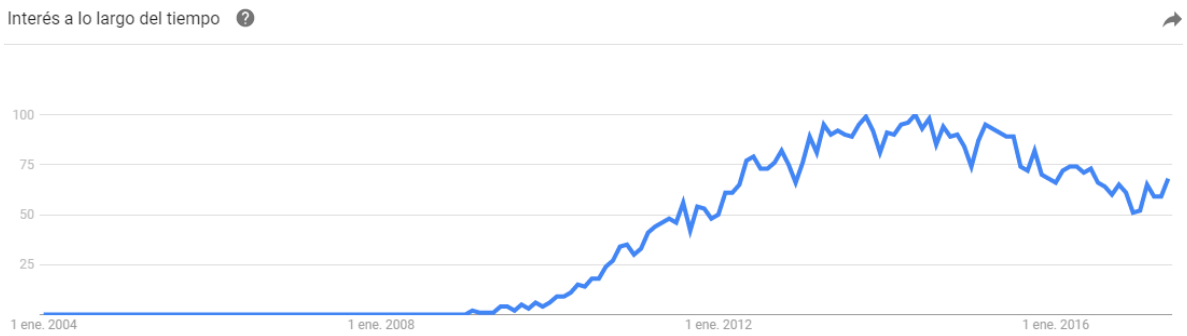
Figura 27: Interés en la búsqueda por la palabra JSF a nivel regional Google Trends



Fuente: Google Trends - <https://trends.google.com/trends/explore?date=all&geo=CO&q=%2Fm%2F026mhl> [citado el 30 de abril de 2017]

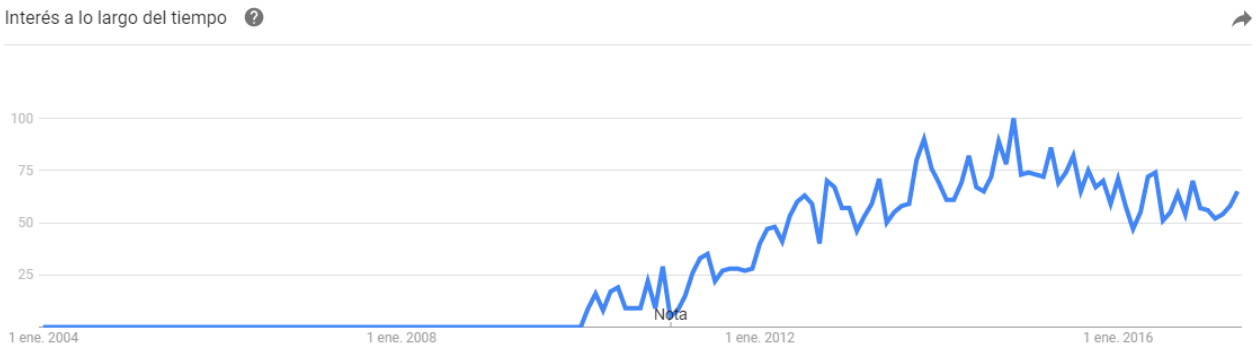
✓ **Búsqueda por la frase JSF Primefaces**

Figura 28: Interés en la búsqueda por la palabra JSF Primefaces a nivel mundial Google Trends



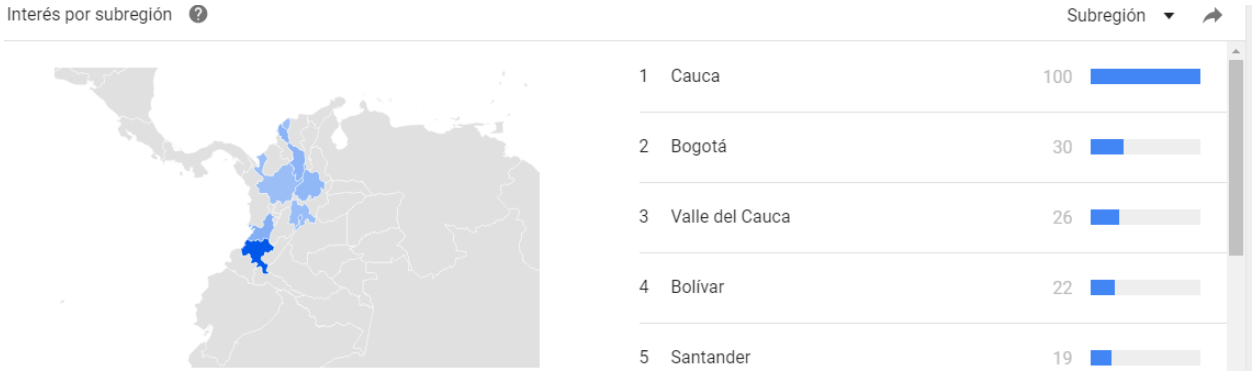
Fuente: Google Trends - <https://trends.google.com/trends/explore?date=all&geo=CO&q=%2Fm%2F026mhl> [citado el 30 de abril de 2017]

Figura 29: Interés en la búsqueda por la palabra JSF primefaces a nivel nacional Google Trends



Fuente: Google Trends - <https://trends.google.com/trends/explore?date=all&geo=CO&q=%2Fm%2F026mhl> [citado el 30 de abril de 2017]

Figura 30: Interés en la búsqueda por la palabra JSF primefaces a nivel regional Google Trends



Fuente: Google Trends - <https://trends.google.com/trends/explore?date=all&geo=CO&q=%2Fm%2F026mhl> [citado el 30 de abril de 2017]

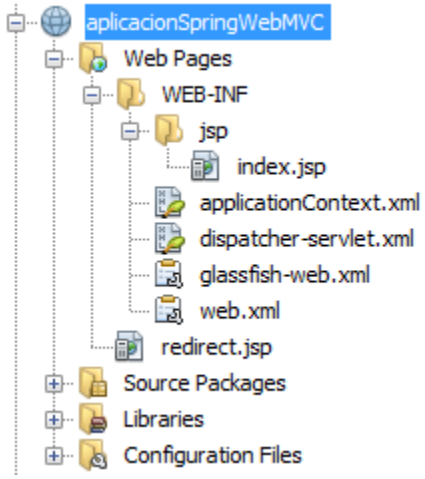
Las gráficas arrojadas por las dos primeras búsquedas tanto a nivel mundial como a nivel nacional muestran un decrecimiento en la web en el interés por JSF sin embargo la última búsqueda que incluye PrimeFaces es todo lo contrario muestra un gran crecimiento en el interés y esto se debe a que fue desarrollada esta nueva tecnología para implementar las interfaces de JSF y resultó muy útil y fácil de usar.

3.2.2 Spring MVC

3.2.2.1 *Funcionalidad.*

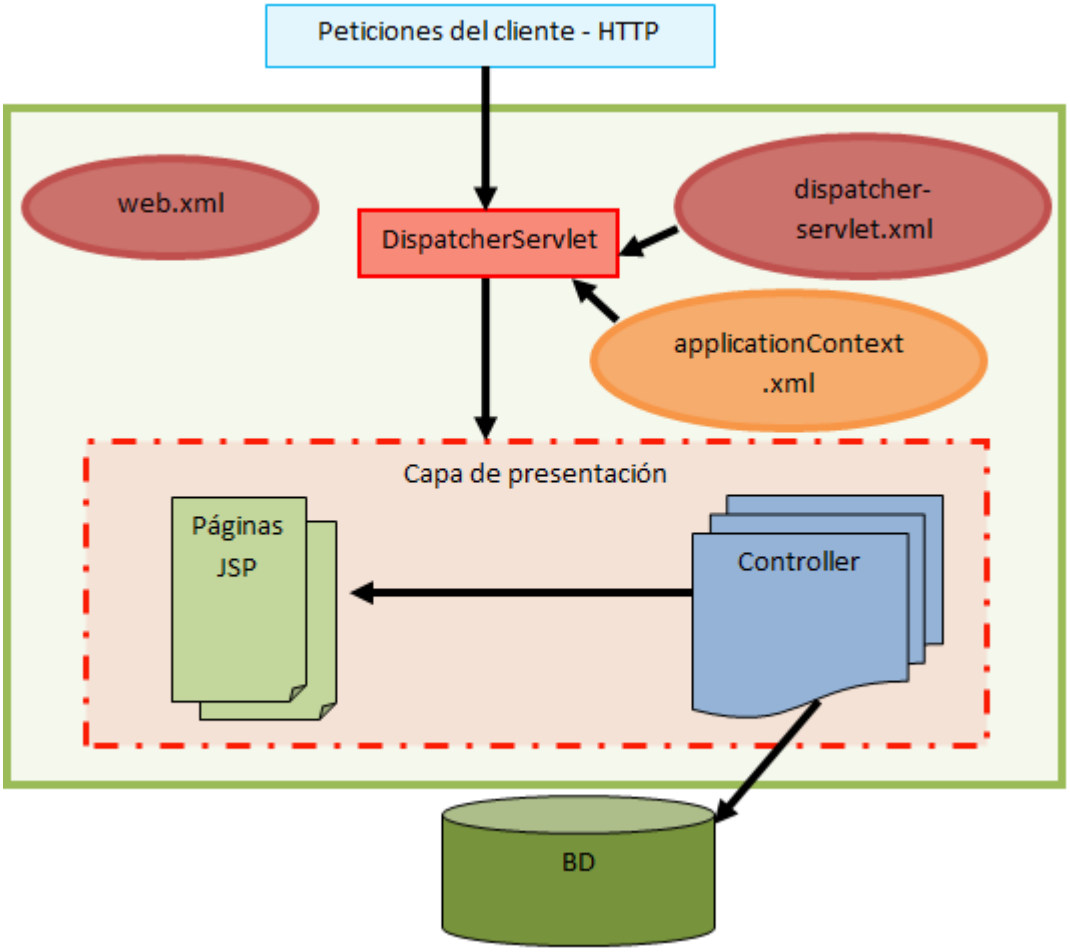
La estructura básica de una aplicación web desarrollada con Spring MVC (ver figura 18) está compuesta por: una carpeta general llamada Web Pages donde está ubicado el archivo **redirect.jsp**, el cual se encarga de redireccionar la aplicación hacia el archivo de inicio **index.jsp** que se encuentra en la subcarpeta jsp (esta carpeta al igual que el archivo index pueden ser renombrados) y ésta a su vez en la sub carpeta WEB-INF donde se crean los archivos **applicationContext.xml**, **dispatcher-servlet.xml**, **glassfish-web.xml** y **web.xml** para la configuraciones del Framework; y una carpeta llamada Source Packages donde se crean los paquetes necesarios que contendrán las clases Java de la aplicación.

Figura 31: Estructura básica aplicación web Spring MVC



Spring MVC a diferencia de JSF tiene un funcionamiento por controladores los cuales son clases Java normales manejados por anotaciones del contenedor. El modelo de funcionamiento de Spring MVC se ve descrito por la siguiente figura.

Figura 32: Modelo de funcionamiento Spring MVC



En la figura 18 se puede observar que las peticiones que le hace el cliente a una aplicación desarrollada bajo el Framework Spring, son controladas por la clase DispatcherServlet quien se encarga de administrar los recursos de toda la aplicación web. A continuación se detallará el funcionamiento de cada uno de los elementos que participan en el sistema descrito.

Archivo de configuración web.xml.

Figura 33: Archivo de configuración web.xml en Spring MVC

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                             http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/applicationContext.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>*.htm</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>redirect.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

La configuración del archivo web.xml en Spring varía con respecto a JSF en solo algunos aspectos ya que el propósito es el mismo, solo que cambia el nombre que se le asigna al servlet que en este caso es **dispatcher**, la clase que lo contiene que en este caso es **DispatcherServlet**, el patrón de las urls que para este caso es ***.htm** y el archivo de bienvenida que se establece como **redirect.jsp**. Tal vez la mayor diferencia es en la etiqueta **listener**, la cual se encarga de hacer el llamado a la clase **ContextLoaderListener**, que se encuentra ubicada en el paquete org.springframework.web.context, para que cargue e inicialice todo el contexto del Framework Spring en el servidor y el archivo de configuración **aplicationContext.xml** que sirve para personalizar el contexto de la aplicación, este archivo es opcional ya que si no se configura se tomará la configuración por defecto, pero en caso que la aplicación a

desarrollar deba establecer conexión con una base de datos debe configurarse estableciendo los parámetros de conexión en su contenido.

La clase DispatcherServlet.

Es la clase principal o el HttpServlet implementado propiamente por Spring MVC y se encarga del control de peticiones hechas a la aplicación, Spring posee un contenedor propio que sostiene todos sus métodos de fábrica y utiliza el principio de inversión de control IoC o inyección de dependencia DI para controlar las clases bean, esto lo hace por medio del uso de anotaciones especificadas al inicio de cada clase o método.

Archivo de configuración dispatcher-servlet.xml.

En este archivo de configuración (ver figura 21) se le especifica al DispatcherServlet de Spring que propiedades específicamente debe implementar para la aplicación. Las páginas que se van a utilizar en la aplicación pueden ser agregadas estáticamente en este archivo para que el DispatcherServlet las cargue en el contexto y puedan ser accedidas, pero sin embargo de no hacerlo así, el DispatcherServlet las interpretará y cargará en el contexto si son instanciadas desde un controlador administrado por la anotación **@Controller**.

Figura 34: Archivo de configuración dispatcher-servlet.xml

```xml
<?xml version='1.0' encoding='UTF-8' ?>
<!-- was: <?xml version="1.0" encoding="UTF-8"?> -->
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/aop http://www.springframework.org/schema/tx
       http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
       http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
       http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">

    <bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping"/>

    <!--
    Most controllers will use the ControllerClassNameHandlerMapping above, but
    for the index controller we are using ParameterizableViewController, so we must
    define an explicit mapping for it.
    -->
    <bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>
                <prop key="index.htm">indexController</prop>
            </props>
        </property>
    </bean>

    <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver"
          p:prefix="/WEB-INF/jsp/"
          p:suffix=".jsp" />

    <bean name="indexController"
          class="org.springframework.web.servlet.mvc.ParameterizableViewController"
          p:viewName="index" />

</beans>
```

En este archivo de configuración se especifican una serie de esquemas como propiedades de la etiqueta **beans**, que además es la raíz del documento, los cuales sirven para validar la estructura de lo que se escribe dentro de cada etiqueta.

bean class="org.springframework.web.servlet.mvc.support.Controller ClassNameHand lerMapping" esta línea de código indica que la clase **ControllerClassNameHand lerMapping** debe encargarse de manejar el mapeo de las url que se llamen desde los controladores.

La etiqueta bean con id=**urlMapping** lo que hace es asignarle el trabajo a la clase SimpleUrlHandlerMapping de tomar el recurso que está solicitando el cliente al servidor y dirigirlo al controlador especificado, en este caso **indexController**, quien será el encargado de crear y mostrar la vista con la información requerida en la url solicitada. De esta manera pueden especificarse con la etiqueta **prop** estáticamente, qué urls serán administradas por qué controladores, escribiendo en su propiedad **key** la url y como argumento dentro de la etiqueta el nombre del controlador que la construirá, mostrará y administrará.

La etiqueta bean con propiedad id=**viewResolver** lo que hace es decirle a la clase InternalResourceViewResolver que se encargue de decirle al DispatcherServlet que interprete todos los llamados a recursos que se encuentran en la ubicación **/WEB-INF/jsp/** con extensión .jsp, con el propósito de evitar escribir la extensión cada vez que se requiera cargar los archivos desde un controlador para devolver una vista.

Como se puede observar en el **name** de la última etiqueta bean descrita en el documento, el nombre coincide con la descripción de la etiqueta **prop** en el bean con id urlMapping, esto es porque la clase **ParameterizableViewController** es la encargada de devolver la vista especificada en la propiedad **viewName.**

Capa de presentación.

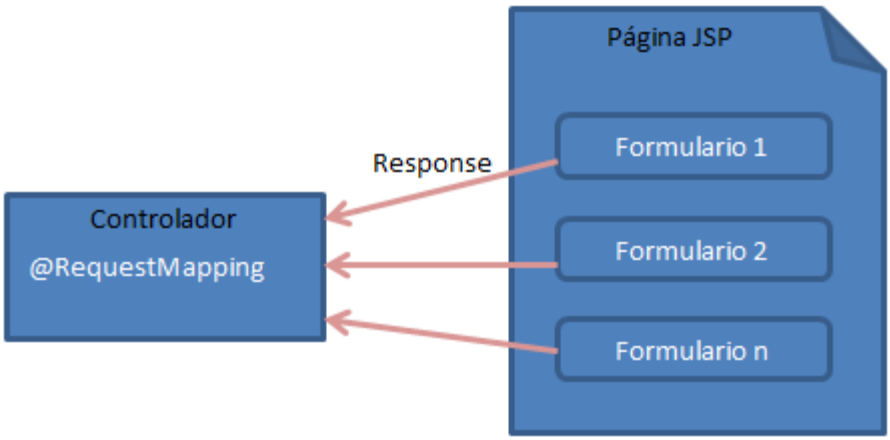
Controladores (Controllers. Los controladores como en cualquier aplicación MVC y como su nombre lo indica son los que se encargan de controlar la interacción entre las peticiones del cliente (por métodos GET y POST) y los recursos disponibles en

el servidor, la diferencia en Spring MVC radica en que Los controladores hacen parte de una capa oculta detrás de la presentación de cada página que recibe dichas peticiones y las maneja por medio de anotaciones como @Controller que sirve para registrar el controlador en el servlet de Spring, @RequestMapping que sirve para relacionar un método del controlador con una petición realizada por el cliente HTTP por medio de formularios (ver figura 22), @RequestParam sirve para recuperar valores de parámetros establecidos en el cliente HTTP por medio de campos de formularios y utilizarlos en el controlador, entre muchas anotaciones que se utilizan en los controladores de Spring pero todas con el fin de mantener una interacción entre el cliente y el servidor.

Páginas JSP. Las páginas JSP no son más que código HTML combinado con código java que se ejecutan en el servidor con un prug-in que les da un manejo especial para que las instrucciones Java se produzcan del lado del cliente logrando así una transformación simultánea.

Las Vistas de Spring MVC se construyen con páginas JSP las cuales interactúan con los controladores como se explicaba anteriormente por medio de formularios que envían datos bien sea por método POST o GET para que el controlador haga algo con ellos y devuelva una respuesta que será visualizada por el usuario.

Figura 35: Interacción entre vista y Controlador en Spring



3.2.2.2 Fiabilidad.

Spring MVC tiene implementado un Framework exclusivamente para manejo de seguridad web llamado Spring Security el cual se analizará a continuación para entender cómo es su funcionamiento en cuanto a manejo de autenticación y autorización durante la navegación.

Autenticación.

Declaración de espacio de nombre. Lo primero que se debe especificar para que Spring Security sea el encargado de la seguridad es el espacio de nombres en el archivo de configuración **dispatcher-servlet** agregando las líneas resaltadas a continuación:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:security="http://www.springframework.org/schema/security"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
       http://www.springframework.org/schema/security/spring-security.xsd
       http://www.springframework.org/schema/security">
    ...
</beans>
```

Declaración de escuchador y filtro. Para empezar, Spring Security controla el inicio de sesión de usuarios agregando al archivo de configuración web.xml la clase escuchadora de eventos **HttpSessionEventPublisher** y la clase **DelegatingFilterProxy** que es una implementación de un filtro propio de Spring que viene siendo un Bean en el contexto de la aplicación.

La clase **HttpSessionEventPublisher** se declara de la siguiente forma:

```
<listener>
        <listener-class>
                org.springframework.security.web.session.HttpSessionEventPublisher
        </listener-class>
</listener>
```

Y la clase **DelegatingFilterProxy** se declara de la siguiente manera:

```
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
     <filter-name>springSecurityFilterChain</filter-name>
     <url-pattern>/*</url-pattern>
</filter-mapping>
```

En esta declaración lo que se hace es especificarle al Servlet que el Bean denominado como **springSecurityFilterChain** controlado por la clase **DelegatingFilterProxy** ubicada en el paquete **org.springframework.web.filter**, va a ser el encargado de manejar la seguridad web de la aplicación. Este Bean es creado por el espacio de nombres declarado al principio.

Declaración de página de login y logout. Es probable que personas con mucho conocimiento en ataques informáticos puedan engañar al servidor enviando datos de logueo desde una página diferente a la original, para evitar que este tipo de situaciones se presenten Spring Security permite especificar en el archivo web.xml, cuál será la única página autorizada para logueo de usuario por medio de la siguiente declaración.

```
<http pattern="/login.html" security="none"/>
<http>
    <intercept-url pattern="/**" access="ROLE_USER, ROLE_ADMIN" />
    <form-login login-page=”/login.htm”
                default-target-url=”/home.htm”
                always-use-default-target=”true” />
     <logout logout-url =”/logout” logout-success-url="/logout.htm"/>
</http>
```

Se le indica al servidor en el primer elemento **http** por medio del atributo **pattern** que la página **login.htm** no estará protegida **security=”none”** ya que esta debe ser accedida por cualquiera.

En el segundo elemento **http** se indica cuál será la url de logueo de usuario por medio del atributo **login-page** en el elemento **form-login**; también se le indica por medio de los atributos **default-target-url** y **always-use-default-target** establecido en true, cuál será la página de inicio, es decir, la url a la que siempre será re-direccionado el usuario luego de loguearse correctamente; el atributo **logout-url** indica hacia donde debe ser re-dirigido el usuario en caso de proporcionar mal las credenciales de inicio de sesión.

Y por último en el elemento **intercept-url** se le indica al servidor que intercepte todas las peticiones a urls de la aplicación (**pattern="/**"**) que tengan acceso restringido

solo para los roles especificados (**access="ROLE_USER, ROLE_ADMIN"**) y re-direccione hacia la página de login.htm.

Control de sesiones. Para evitar que un usuario inicie sesión más de una vez se debe agregar al contexto de la aplicación las siguientes líneas:

```
<http>
    <session-management session-authentication-error-url=”/url de error”>
       <concurrency-control max-sessions="1" error-if-maximum-exceeded="true" />
    </session-management>
 </http>
```

Esto además de impedir que el usuario inicie sesión más de una vez lo re-direcciona a una url de error especificada en el atributo **session-authentication-error-url** del elemento **session-management.**

Autorización.

Configuración estática. Se puede configurar globalmente desde el archivo de configuración web.xml el llamado o ejecución a métodos de Beans administrados de la siguiente forma:

```
 <global-method-security>
    <protect-pointcut
            expression="execution(* ActualizarUsuarios(..))"
            access=" Administrador "/>
</global-method-security>
```

Lo anterior establece que se quiere interceptar el llamado o ejecución de cualquier método que exista en la aplicación que se llame ActualizarUsuarios independientemente

de la cantidad de argumentos que tenga e independientemente del valor que retorne y garantizar que solo se ejecute bajo el rol Administrador.

Configuración por anotaciones. Para utilizar las anotaciones de Spring Security en una aplicación el primer paso es establecer en el archivo de configuración web.xml la siguiente línea:

<global-method-security pre-post-annotations="enabled"/>

Todos los objetos de autenticaciones realizadas al principio son almacenados en la clase **GrantedAuthority** propia de Spring Security y estos objetos son recuperados por la clase Authentication por medio de la clase **AuthenticationManager** y leídos por medio de la clase **AccessDecisionManager** al momento de tomar decisiones cuando se invocan métodos o se hacen peticiones web.

Las anotaciones más utilizadas en Spring Security son @PreAuthorize, @PreFilter, @PostAuthorize y @PostFilter de estas cuatro la más importante de analizar teniendo en cuenta el tema de autorización es @PreAuthorize la cual sirve para comprobar la autorización antes de ser ejecutado cualquier método de Bean administrado que tenga establecida dicha anotación, para su buen funcionamiento se debe agregar por medio de lenguaje SpEL el argumento necesario donde se le especifican las condiciones que deben cumplirse para poder ejecutar dicho método.

Por ejemplo si se tiene dos roles establecidos Administrador y Usuario y se tiene un Bean llamado usuarioBean que posee dos métodos Listar y Actualizar información de

usuarios del sistema. El método Listar puede ser accedido por cualquier rol y el método Actualizar solo puede ser accedido por el rol Administrador, por medio de anotaciones se vería la restricción de la siguiente forma:

```
public class usuarioBean (){

    @ PreAuthorize("hasRole('Administrador') AND hasRole('Usuario') ")
    public …  Listar(…){
          …
         }

    @PreAuthorize("hasRole('Administrador') ")
    public … <usuarios> Actualizar(…){
          …
         }
}
```

Es decir que si un usuario con rol Usuario está logueado no tendrá acceso al método Actualizar ya que no tiene autorización para hacerlo.

Las expresiones que se pueden utilizar dentro de las anotaciones @PreAuthorize, @PostAuthorize, @PreFilter y @PostFilter se describen en la siguiente tabla.

Tabla 9: Expresiones para la anotación @PreAuthorize en Spring Security

Expresión	Descripción
hasRole([role])	Devuelve “true” si el rol actual es el especificado en el argumento de la función. Predeterminadamente agrega ROLE_ al inicio del argumento pero esto puede ser modificado y personalizado en el defaultRolePrefix de la clase DefaultWebSecurityExpressionHandler
hasAnyRole([role1,role2])	Retorna “true” si el rol actual concuerda con alguno de los especificados en el argumento de la función (los argumentos son pasados como una lista de caracteres separados por comas) si los argumentos no inician por ROLE_ se lo asignará automáticamente, también puede

	ser modificado en defaultRolePrefix.
hasAuthority([authority])	Cuando se configura autoridad para un rol esta expresión devuelve “true” si esa autoridad del rol actual coincide con la especificada en el argumento de la función.
hasAnyAuthority([authority1, authority2])	Devuelve “true” si la autoridad actual coincide con alguna de las autoridades especificadas en el argumento de la función
principal	Permite acceso directo al objeto principal que representa al usuario actual
authentication	Permite acceso directo al objeto de autenticación actual GrantedAuthority obtenido de la clase SecurityContext
permitAll	Permite acceso a todos los roles
denyAll	Deniega acceso a todos los roles
isAnonymous()	Retorna “true” si el usuario actual es anónimo
isRememberMe()	Retorna “true” si el usuario está en memoria
isAuthenticated()	Devuelve “true” si el usuario no es anonimo
isFullyAuthenticated()	Devuelve “true” si el usuario no es anónimo y tampoco está en memoria
hasPermission(Object target, Object permission)	Devuelve true si el usuario tiene acceso al destino proporcionado para el permiso dado. Por ejemplo, hasPermission (BeanUsuario, 'actualizar')
hasPermission(Object targetId, String targetType, Object permission)	Devuelve true si el usuario tiene acceso al destino proporcionado para el permiso dado.

3.2.2.3 Mantenibilidad

Spring MVC es un proyecto que inició en 2002 por el programador Rod Johnson quien además es el fundador de la organización Spring Source. La primera versión fue la 1.0 lanzada en marzo de 2004 con algunas mejoras hechas el mismo año, sin embargo en

el sitio oficial de Spring MVC solo se encuentra documentación correspondiente a su versión 3.2.18 hasta su versión 5.0.0 que actualmente está en RC .

En la siguiente tabla se describen las versiones de Spring que tienen soporte actualmente por su organización;

Tabla 10: Soporte para versiones de Spring MVC

Versión	Fecha de lanzamiento	Web de documentación	Web de API
3.2.18	Noviembre de 2013	http://docs.spring.io/spring/docs/3.2.18.RELEASE/spring-framework-reference/htmlsingle/	http://docs.spring.io/spring/docs/3.2.18.RELEASE/javadoc-api/
4.2.9	Diciembre de 2016	http://docs.spring.io/spring/docs/4.2.9.RELEASE/spring-framework-reference/htmlsingle/	http://docs.spring.io/spring/docs/4.2.9.RELEASE/javadoc-api/
4.3.8	Abril de 2017	http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/	http://docs.spring.io/spring/docs/current/javadoc-api/
4.3.9	Abrir de 2017	http://docs.spring.io/spring/docs/4.3.9.BUILD-SNAPSHOT/spring-framework-reference/htmlsingle/	http://docs.spring.io/spring/docs/4.3.9.BUILD-SNAPSHOT/javadoc-api/
5.0.0.M5	Febrero de 2017	http://docs.spring.io/spring/docs/5.0.0.BUILD-SNAPSHOT/spring-framework-reference/htmlsingle/	http://docs.spring.io/spring/docs/5.0.0.BUILD-SNAPSHOT/javadoc-api/

Spring MVC en su sitio web ofrece guías de aprendizaje para usuarios desarrolladores en la siguiente tabla se relacionan algunas con el fin de ver el gran soporte que le da su organización.

Tabla 11: Guías de soporte para manejo de Spring MVC

Guía	Descripción	Web
Spring Security Architecture	Guía sobre la arquitectura de Spring Security, contiene explicación sobre cómo es el manejo y control de autenticación y autorización en aplicaciones web.	https://spring.io/guides/topicals/spring-security-architecture/
Building a RESTful Web Service	Guía para aprender a crear un servicio web RESTful.	https://spring.io/guides/gs/rest-service/
Consuming a RESTful Web Service	Guía para aprender a consumir servicios web RESTful por medio de recuperación de datos de páginas web utilizando RestTemplate.	https://spring.io/guides/gs/consuming-rest/
Authenticating a User with LDAP	Guía que contiene información sobre cómo configurar autenticación LDAP en aplicaciones web	https://spring.io/guides/gs/authenticating-ldap/
Validating Form Input	Guía para aprender a realizar validación de formularios con Spring	https://spring.io/guides/gs/validating-form-input/
Securing a Web Application	Guía para aprender a proteger una aplicación web con Spring Security	https://spring.io/guides/gs/securing-web/
Managing Transactions	Guía para aprender a envolver código para el manejo de transacciones	https://spring.io/guides/gs/managing-transactions/
Accessing data with MySQL	Guía para aprender cómo configurar y administrar cuentas de usuario en MySQL y cómo configurar Spring Boot para conectarse a ella en tiempo de ejecución.	https://spring.io/guides/gs/accessing-data-mysql/
Building REST services with Spring	Tutorial para aprender a crear, probar y asegurar fácilmente servicios RESTful con Spring	https://spring.io/guides/tutorials/bookmarks/
Spring Security and Angular JS	Tutorial para aprender cómo usar Spring Security con una aplicación de una sola página con varias arquitecturas de backend, que van desde un simple servidor a una API completa con autenticación OAuth2.	https://spring.io/guides/tutorials/spring-security-and-angular-js/

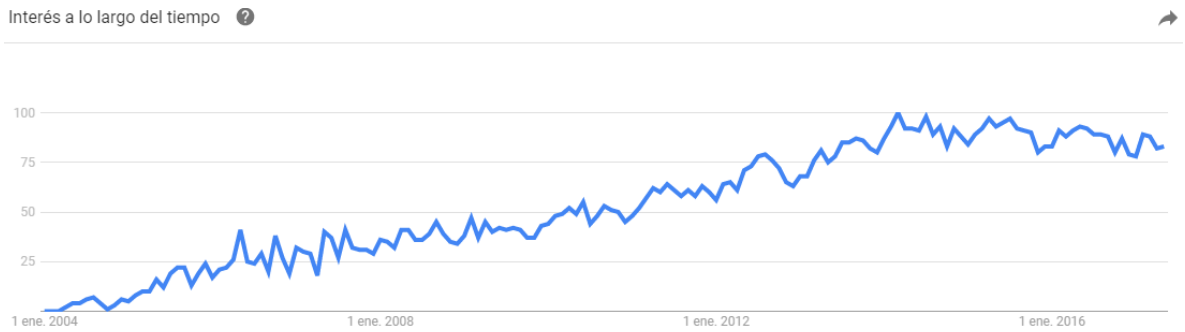
Entre muchos otros tutoriales y guías simples y complejas que dispone la comunidad Spring MVC para los usuarios del mismo y que se pueden encontrar en su sitio oficial https://spring.io/

3.2.2.4 Índice de interés

En esta sección se utilizó la herramienta de estadísticas de búsqueda Google Trends que muestra una línea de tiempo o histórico de búsqueda de una frase o palabra específica, para el caso de estudio se realizó la consulta con dos frases **Spring MVC** y **Spring Framework** a nivel nacional e internacional y los resultados arrojados son los descritos a continuación.

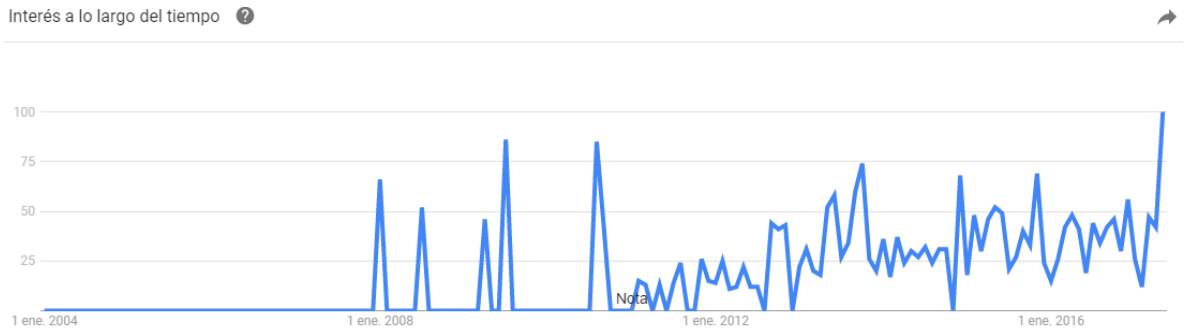
- ✓ **Búsqueda por la frase Spring MVC**

Figura 36: Interés en la búsqueda por la frase Spring MVC a nivel mundial Google Trends



Fuente: Google Trends - <https://trends.google.com/trends/explore?date=all&q=%2Fm%2F026mhl> [citado el 30 de abril de 2017]

Figura 37: Interés en la búsqueda por la frase Spring MVC a nivel nacional Google Trends

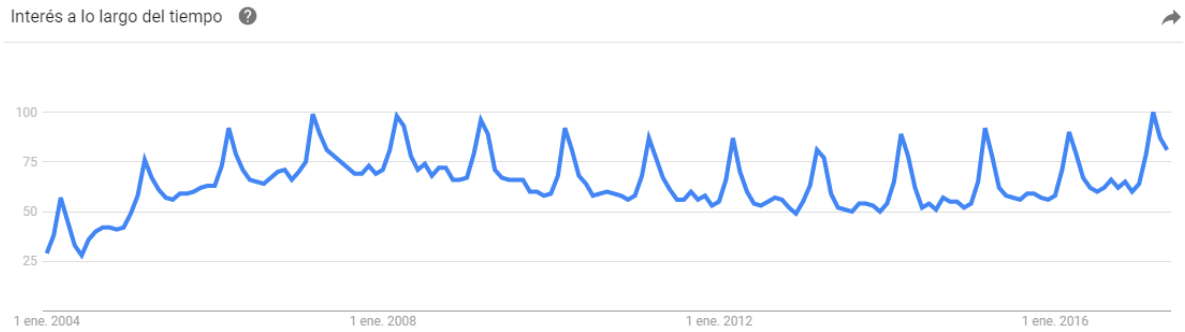


Fuente Google Trends - <https://trends.google.com/trends/explore?date=all&q=%2Fm%2F026mhl> [citado el 30 de abril de 2017]

La búsqueda no arrojó resultados a nivel regional

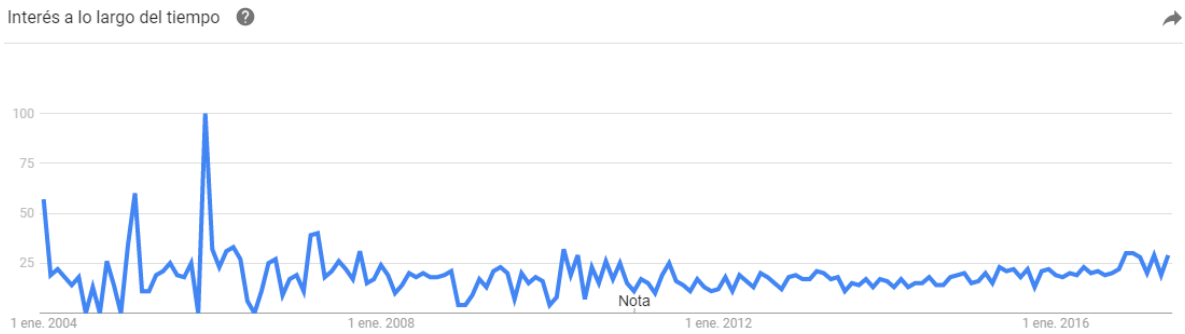
✓ **Búsqueda por la frase Spring Framework**

Figura 38: Interés en la búsqueda por la frase Spring Framework a nivel mundial Google Trends



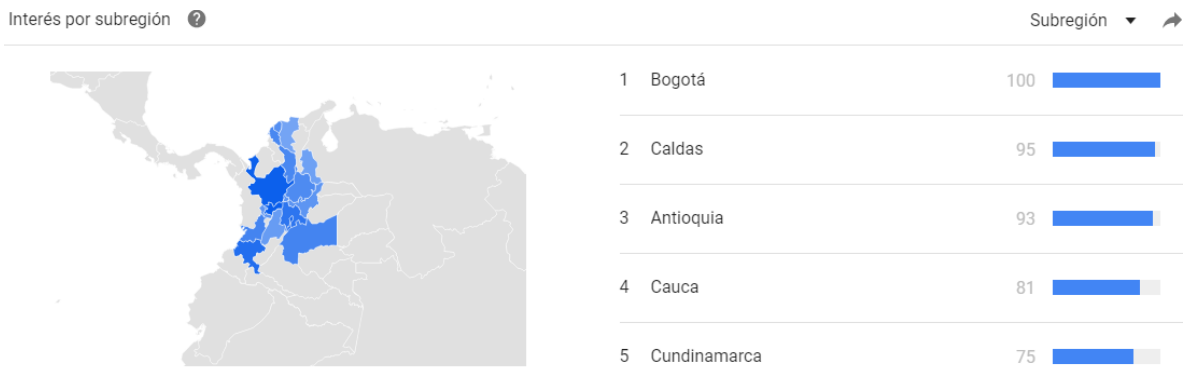
Fuente: Google Trends - <https://trends.google.com/trends/explore?date=all&q=%2Fm%2F026mhl> [citado el 30 de abril de 2017]

Figura 39: Interés en la búsqueda por la frase Spring Framework a nivel nacional Google Trends



Fuente Google Trends - <https://trends.google.com/trends/explore?date=all&q=%2Fm%2F026mhl> [citado el 30 de abril de 2017]

Figura 40: Interés en la búsqueda por la frase Spring Framework a nivel regional Google Trends



Fuente: Google Trends - <https://trends.google.com/trends/explore?date=all&q=%2Fm%2F026mhl> [citado el 30 de abril de 2017]

Los resultados por ambas frases muestran una gráfica ascendente desde el 2004 hasta la actualidad lo que indica que ha ido tomando potencia en la web.

3.3 Selección del Framework web MVC open source

Para la selección del Framework web MVC más apropiado para el propósito del proyecto se ha utilizado como referencia el capítulo Administración de la Fundamentación del libro Ingeniería de Software Orientado a Objetos de Bernd Bruegge y Allen H. Dutoit (Bruegge & Dutoit, 2002), donde se plantea un problema al cuál se le pretende dar solución por medio de alternativas evaluadas según algunos criterios asignando puntuaciones validadas por argumentaciones y al final la alternativa con mayor total de puntuación será la decisión tomada.

El problema. Es escoger el Framework más apropiado para adaptarse a la infraestructura del CIADTI al ser utilizado para transformar el aplicativo Administrador Vortal, la pregunta a resolver es ¿cuál es el Framework web MVC open source más apropiado para la transformación del aplicativo Administrador Vortal en el CIADTI?

Las alternativas de solución. Son los Frameworks tratados anteriormente JavaServer Faces y Spring MVC

Los criterios de evaluación. Son funcionalidad, fiabilidad, mantenibilidad e índice de interés en el tiempo descrito por gráficas de la herramienta Google Trends.

Las calificaciones estarán entre el rango de 1 a 5 y el argumento que evalúa cada una se describe en la siguiente tabla.

Tabla 12: Rango para calificación de JSF Y Spring MVC

Calificación	Descripción
1	El Framework no cumple con el criterio
2	El Framework cumple con menos del 40% del criterio
3	El Framework está cumpliendo entre el 40% y 69% del criterio
4	El Framework cumple entre 70% y 89% del criterio
5	El Framework cumple entre 90% y 100% del criterio

Tabla 13: Matriz de calificación y justificación de selección entre JSF y Spring MVC

Criterios de selección	Alternativas de solución	
	JavaServer Faces	Spring MVC
	Calificación y Justificación	
Funcionalidad	Calificación 5 Porque para facilitar la experiencia del usuario proporciona un amplio conjunto de componentes para la construcción de la interfaz gráfica de sus aplicaciones web como son: * Clases propias para mejoramiento del funcionamiento de los elementos estándares de HTML que son muy simples de utilizar. * Clases JavaBeans que se encargan de controlar la interfaz enviando eventos del lado del cliente al servidor y viceversa de forma muy sencilla. * Clases PrimeFaces que incluyen más componentes para construcción de la vista e incluye integración con Ajax y funciones JavaScript para manejar eventos de forma muy sencilla. * Fácil Integración de etiquetas	Calificación 4 En Spring MVC se ha alcanzado un gran nivel de funcionamiento en cuanto a interacción entre las páginas y los controladores logrando que las peticiones cliente-servidor sean más fluidas manteniendo el patrón vista controlador, sin embargo aún sigue utilizando páginas JSP para la construcción de la vista e interacción con el usuario y aunque permite la integración con JSF para este propósito no ha construido su propio lenguaje para construcción de interfaz gráfica.

	RichFaces e IceFaces para construcción e la vista.	
Fiabilidad	Calificación 5 Maneja un estándar JAAS (Java Authentication and Authorization Service) clases e interfaces de Java para control de autenticación y autorización como: * La clase LoginContext que sirve para proporcionar los métodos necesarios para manejo de autenticación de usuarios. * La interfaz LoginModule ofrece la variabilidad para implementar diferentes tipos de tecnologías para autenticación en una aplicación. * La interfaz CallbackHandler diseñada para pasar al LoginContex la información de autenticación de usuarios. * AuthPermission Clase para encapsular los permisos de usuarios * La clase PrivateCredentialPermission protege el acceso a las credenciales privadas de cada usuario en cualquier momento. Todo un sistema de control para autenticación y autorización disponible para las aplicaciones JSF por ser estándar de JEE.	Calificación 5 Su organización desarrolladora se ha preocupado por la implementación de una API especializada en seguridad web, la cual es Spring Security, que hace un control muy completo de autenticación y autorización de usuarios para garantizar integridad de credenciales. Spring maneja clases como: HttpSessionEventPublisher, DelegatingFilterProxy, springSecurityFilterChain, GrantedAuthority, AuthenticationManager, entre otras dedicadas a asegurar la aplicación.
Mantenibilidad	Calificación 5 Los procesos de mantenimiento de JSF son constantes e incluyen mejoras significativas, todas con el fin de volver más agradable la experiencia del usuario y por ser estándar de JEE están sometidos al control de Java Specification	Calificación 4 Spring MVC es un proyecto que se puso en funcionamiento desde marzo de 2004 y su primera versión fue la 1.0 sin embargo no contiene en su sitio oficial documentación correspondiente a versiones anteriores a la 3.2, adicional a esto el tiempo

	Request lo que implica mayor confiabilidad en el proceso.	transcurrido entre el lanzamiento de la versión 3.2.18 y la versión 4.2 es de 3 años y el tiempo transcurrido entre la versión 4.2 y la versión 4.3 es de un año y sus mejoras de mantenimiento han sido limitadas. La versión 5.0 está en prueba actualmente lo que indica que en los dos últimos años ha sido más constante el soporte y a futuro promete ir mejorando.
Índice de interés	Calificación 4 * En impacto al momento de ser lanzado mostró un índice de interés muy alto tanto así que entre 2007 y 2008 llegó a 100%. * La permanencia de interés fue disminuyendo tanto a nivel mundial como a nivel nacional a partir de 2008. * Entre 2009 y 2010 se incluyó el proyecto PrimeFaces y desde entonces ha ido aumentando el interés por la integración con JSF. Por la persistencia de la comunidad a cargo del Framework JSF e incluir nuevos componentes para su mejora como PrimeFaces, IceFaces y RichFaces que incrementan el interés de la comunidad desarrolladora de aplicaciones web y mejorar constantemente los ya existentes es que recibe la calificación asignada.	Calificación 4 * Su impacto al momento de su lanzamiento muestra un índice de interés bajo en el 2004 y se mantiene hasta el 2012. * A partir de 2012 se observa un aumento constante en el interés por el Framework Spring MVC a nivel mundial sin embargo a nivel nacional no arrojó ningún dato la búsqueda lo que indica que el interés en Colombia no es muy alto. * La organización de Spring MVC se ha empeñado en los últimos dos años a hacer mejoras significativas para atraer más desarrolladores a utilizarlo y lo ha logrado pero sin embargo por su necesidad de cambiar la construcción de las páginas con a otro entorno diferente a JSP no le permite ser mejor que JSF en cuanto a facilidad de uso y por lo tanto le resta interés por parte de los usuarios.
Total	19 Obtuvo un total de diecinueve puntos en su calificación	17 Obtuvo un total de diecisiete puntos en su calificación

La matriz de decisión arrojó una calificación de diecinueve puntos para el Framework JavaServer Faces y una calificación de diecisiete puntos para el Framework Spring MVC por lo

tanto la decisión es seleccionar a JSF para el próximo capítulo hacer el prototipo de aplicación poniendo a prueba sus funcionalidades.

3.4 Construcción del prototipo

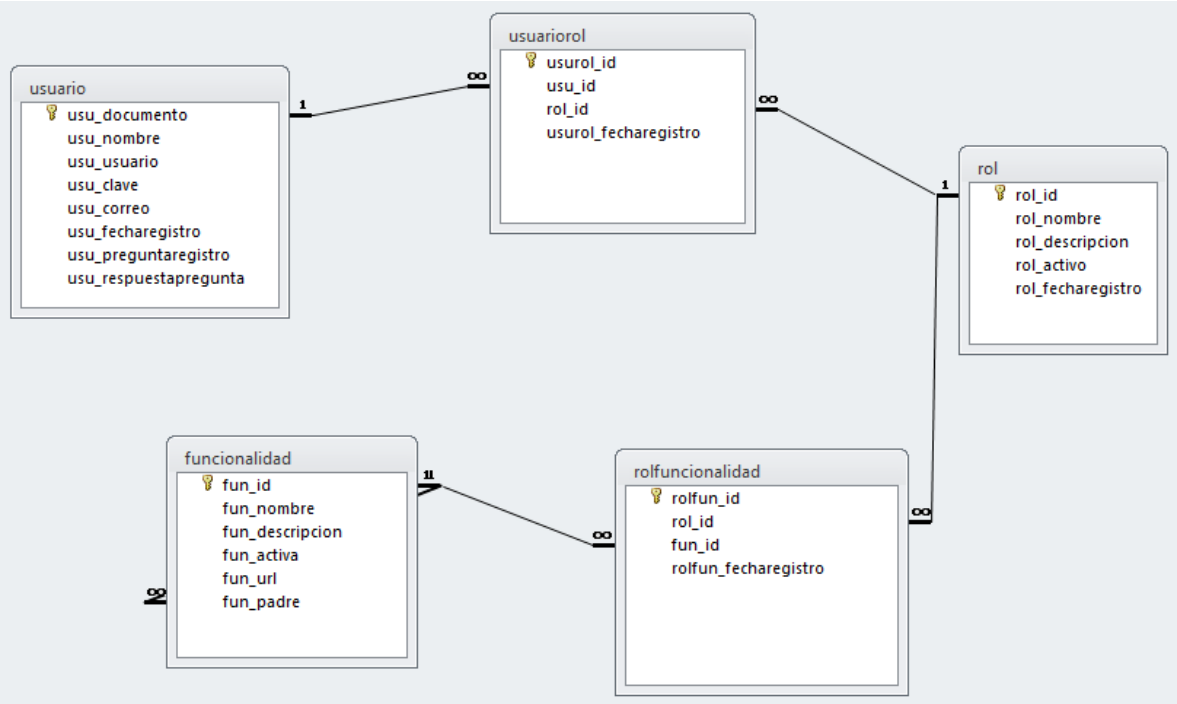
Esta sección contiene explicación sobre el proceso de desarrollo del prototipo de software en JSF y prueba de algunas funcionalidades del mismo, contiene un análisis sobre los resultados observados durante el proceso y por ultimo las conclusiones del proyecto.

Este proceso consiste en crear una base de datos con las tablas usuario, rol y funcionalidad con sus respectivas relaciones usuariorol y rolfuncionalidad. A partir de esta base de datos, las funcionalidades a tener en cuenta son el logueo de usuarios, la asignación de roles a usuarios y la asignación de funcionalidades por rol.

El proceso se dividirá en dos partes que son la construcción del modelo, la construcción e interacción entre la vista los controladores.

3.4.1 El modelo

Figura 41: Modelo entidad relación de la base de datos



La API de Hibernate transforma este modelo a uno orientado a objetos convirtiendo cada entidad en un objeto para ser accedidos desde los controladores.

Figura 42: Modelo objeto relacional

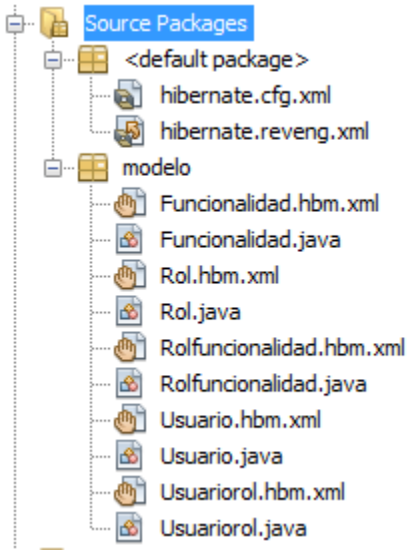


Figura 43: Archivo de mapeo de la entidad usuariorol

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<!-- Generated 24/05/2017 08:43:13 AM by Hibernate Tools 4.3.1 -->
<hibernate-mapping>
    <class name="modelo.Usuariorol" table="usuariorol" schema="vortal"
        optimistic-lock="version">
        <id name="usurolId" type="int">
            <column name="usurol_id" />
            <generator class="assigned" />
        </id>
        <many-to-one name="rol" class="modelo.Rol" fetch="select">
            <column name="rol_id" not-null="true" />
        </many-to-one>
        <many-to-one name="usuario" class="modelo.Usuario" fetch="select">
            <column name="usu_id" not-null="true" />
        </many-to-one>
        <property name="usurolFecharegistro" type="timestamp">
            <column name="usurol_fecharegistro" length="22" not-null="true" />
        </property>
    </class>
</hibernate-mapping>
```

Figura 44: El objeto usuariorol

```
@Entity
@Table(name="usuariorol"
    ,schema="vortal"
)
public class Usuariorol  implements java.io.Serializable {


     private int usurolId;
     private Rol rol;
     private Usuario usuario;
     private Date usurolFecharegistro;

    public Usuariorol() {
    }

    public Usuariorol(int usurolId, Rol rol, Usuario usuario, Date usurolFecharegistro) {
       this.usurolId = usurolId;
       this.rol = rol;
       this.usuario = usuario;
       this.usurolFecharegistro = usurolFecharegistro;
    }

     @Id


     @Column(name="usurol_id", nullable=false)
     public int getUsurolId() {
         return this.usurolId;
     }

     public void setUsurolId(int usurolId) {
         this.usurolId = usurolId;
     }

@ManyToOne(fetch=FetchType.LAZY)
     @JoinColumn(name="rol_id", nullable=false)
     public Rol getRol() {
         return this.rol;
     }

     public void setRol(Rol rol) {
         this.rol = rol;
     }

@ManyToOne(fetch=FetchType.LAZY)
     @JoinColumn(name="usu_id", nullable=false)
     public Usuario getUsuario() {
         return this.usuario;
     }

     public void setUsuario(Usuario usuario) {
         this.usuario = usuario;
     }

     @Temporal(TemporalType.TIMESTAMP)
     @Column(name="usurol_fecharegistro", nullable=false, length=22)
     public Date getUsurolFecharegistro() {
         return this.usurolFecharegistro;
     }

     public void setUsurolFecharegistro(Date usurolFecharegistro) {
         this.usurolFecharegistro = usurolFecharegistro;
     }
```

Así mismo se construyen cada una de las clases que representan las entidades de la base de datos de esta forma Hibernate permite persistir datos sencillamente desde los controladores.

3.4.2 La vista y controlador

En esta sección el propósito es evidenciar algunas de las funcionalidades del prototipo administrador Vortal mostrando la forma como interactúan las vistas con los controladores.

3.4.2.1 Internacionalización

Para la manejo de internacionalización en JSF el primer paso es crear el archivo de propiedades con los nombres de botones, ventanas, cuadros de texto, mensajes y elementos en general utilizados en las páginas del sitio web.

Figura 45: Archivo de configuración espanol_es.properties

```
#////////////////////////////////////////////////////////////////////////////
#mensajes de la página login
#////////////////////////////////////////////////////////////////////////////
loginTitulo                         =   Logueo Usuario Vortal
PanelTitulo                         =   Ingrese usuario y contraseña
OutputLabelUsuario                  =   Usuario
OutputLabelContrasena               =   Contraseña
CommandButtonIniciarSesion          =   Iniciar Sesión

#////////////////////////////////////////////////////////////////////////////


#////////////////////////////////////////////////////////////////////////////
#mensajes de la pagina adminVortal
#////////////////////////////////////////////////////////////////////////////
adminVortalTitulo                   =   Administrador Vortal
...
```

Luego se configura en el archivo faces-config.xml agregando las líneas.

```
<application>
    <locale-config>
```

```
        <default-locale>es</default-locale>
        <supported-locale>en</supported-locale>
        <supported-locale>es_CO</supported-locale>
        <supported-locale>en_US</supported-locale>
      </locale-config>
      <resource-bundle>
        <base-name>internacionalizacion.espanol_es</base-name>
        <var>msg</var>
      </resource-bundle>
    </application>
```

Y por último se utiliza, haciendo referencia a cada etiqueta por su nombre en las páginas por medio de la variable msg declarada anteriormente.

```
<p:panel header="#{msg['PanelTitulo']}">
<h:outputLabel for="username" value="#{msg['OutputLabelUsuario']}"/>
<h:outputLabel for="password" value="#{msg['OutputLabelContrasena']}"/>
<p:commandButton id="loginButton"
value="#{msg['CommandButtonIniciarSesion']}"/>
```

Visualmente se observará como en la figura 46.

3.4.2.2 Inicio de sesión

Figura 46: Página de logueo de usuarios

El usuario ingresa sus datos y presiona el botón iniciar sesión.

Figura 47: Código del formulario de logueo

```
<p:growl id="growl" showDetail="true" life="6000"/>
<h:form id="formLoginVortal" >
    <p:panel header="#{msg['PanelTitulo']}">
        <h:panelGrid columns="2" cellpadding="5">
            <h:outputLabel value="#{msg['OutputLabelUsuario']}"/>
            <p:inputText value="#{loginControl.usuario.usuUsuario}"
                         required="true"
                         id="username"
                         requiredMessage="Debe ingresar nombre de usuario"/>
            <h:outputLabel value="#{msg['OutputLabelContrasena']}"/>
            <p:password value="#{loginControl.usuario.usuClave}"
                        required="true"
                        id="password"
                        requiredMessage="Debe ingresar su contraseña"/>
            <f:facet name="footer">
                <p:commandButton id="loginButton"
                                 value="#{msg['CommandButtonIniciarSesion']}"
                                 update=" :growl"
                                 process="@this, password, username"
                                 action="#{loginControl.autenticar()}"
                                 oncomplete="logueoRequest(xhr, status, args)"/>
            </f:facet>
        </h:panelGrid>
    </p:panel>
</h:form>
```

Los datos son directamente pasados a un Bean de respaldo de la página de logueo llamado loginControl.

Figura 48: Bean de respaldo para la página de logueo (loginControl)

```
@Named(value = "loginControl")
@SessionScoped
public class loginControl implements Serializable {

    private Usuario usuario;

    public Usuario getUsuario() {
        return usuario;
    }

    public void setUsuario(Usuario usuario) {
        this.usuario = usuario;
    }

    public loginControl() {
        usuario = new Usuario();
    }
```

Si el usuario presiona el botón iniciar sesión con los datos correctos. El Bean construye un nuevo objeto usuario con los datos proporcionados y, por medio del atributo value en los elementos inputText y password le pasa los valores utilizando lenguaje EL (value="#{loginControl.usuario.usuUsuario}", #{loginControl.usuario.usuClave}) (ver figura 47) una vez obtenidos estos datos, el atributo action del elemento commandButton hace el llamado al método autenticar quien se encarga de crear la sesión al usuario y enviarlo a la página de selección de rol (ver figura 49).

Figura 49: Construcción de sesión en método autenticar

```
context.addCallbackParam("logueo", true);
ExternalContext externalContex = FacesContext.getCurrentInstance().getExternalContext();
externalContex.getSessionMap().put("usuario", usuario);
externalContex.redirect(externalContex.getRequestContextPath() + "/vistas/seleccionarRol.xhtml");
```

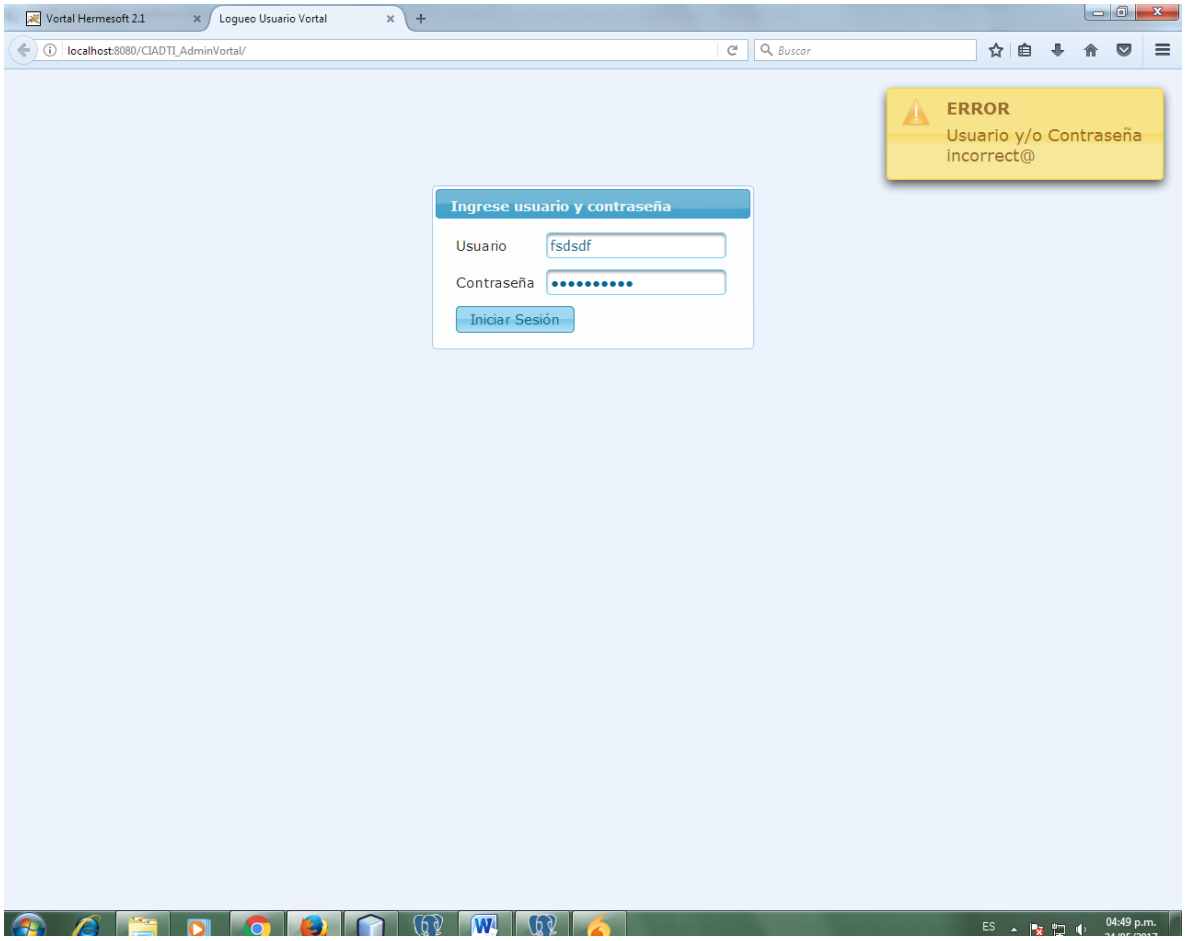
Si el usuario presiona el botón iniciar sesión con datos incorrectos. El método autenticar devuelve un error indicando que el usuario o contraseña son incorrectos.

Figura 50: Devolución de error de logueo del método autenticar

```
FacesMessage msg = new FacesMessage(FacesMessage.SEVERITY_WARN, "ERROR", "Usuario y/o Contraseña incorrect@");
FacesContext.getCurrentInstance().addMessage(null, msg);
context.addCallbackParam("logueo", false);
```

En la página de logueo se recibe el mensaje por medio del elemento growl (ver figura 47) quien se encarga de hacerlo visible en pantalla.

Figura 51: Mensaje de error de logueo en pantalla

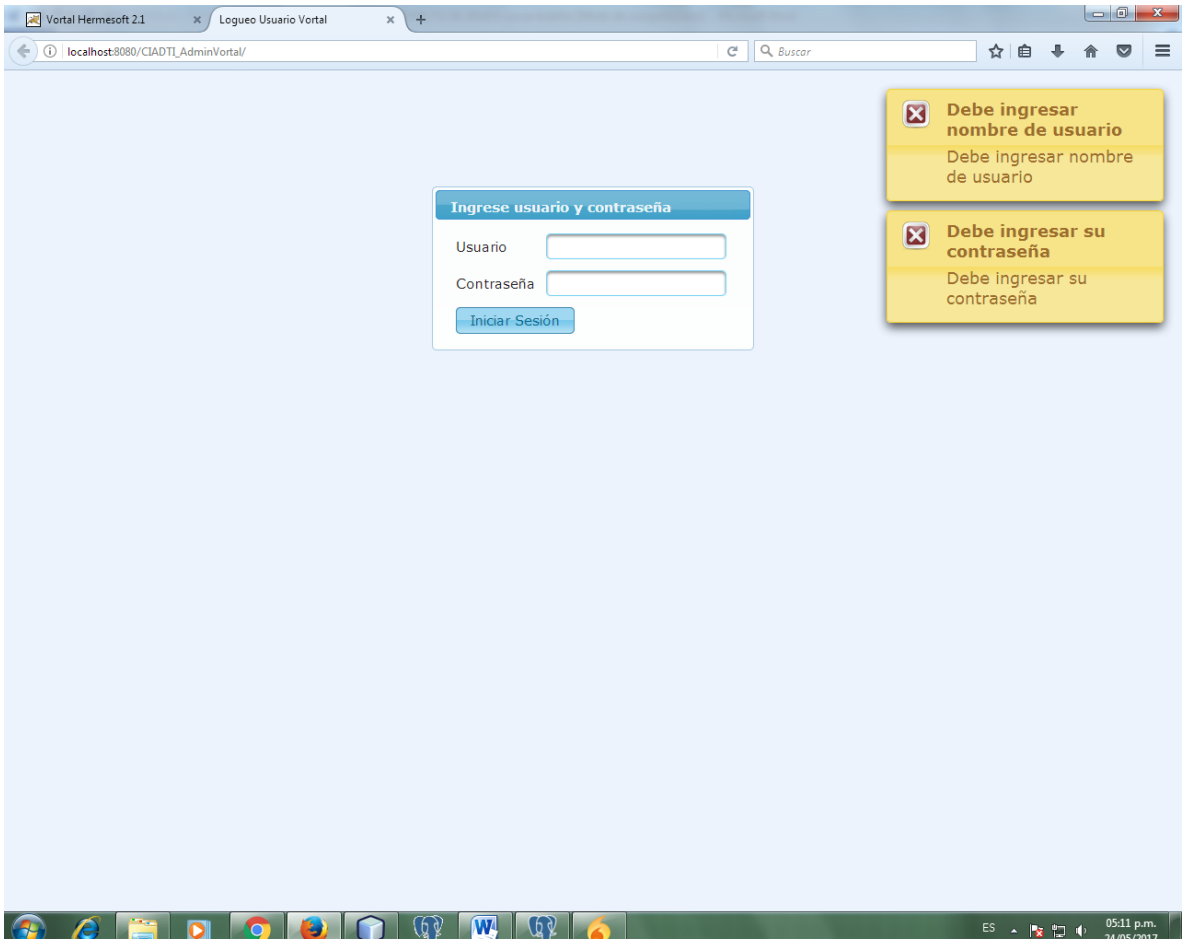


Validación de campos vacíos en logueo. Al momento de presionar el botón iniciar sesión se valida si los campos usuario y contraseña contienen información por medio del atributo process del elemento commandButton donde se escriben los id de los campos que se desean verificar (username y password), en los elementos inputText y

password se asignan dos atributos, required con argumento true para saber que no deben ser enviados sin valor y requiredMessage donde se escribe el mensaje que se verá en pantalla en caso de querer enviar el elemento nulo (ver figura 47).

En pantalla se puede observar que si se intenta enviar el formulario con los elementos vacíos muestra una alerta y no deja continuar con el proceso hasta llenarlos.

Figura 52: Pantalla de error campos vacíos

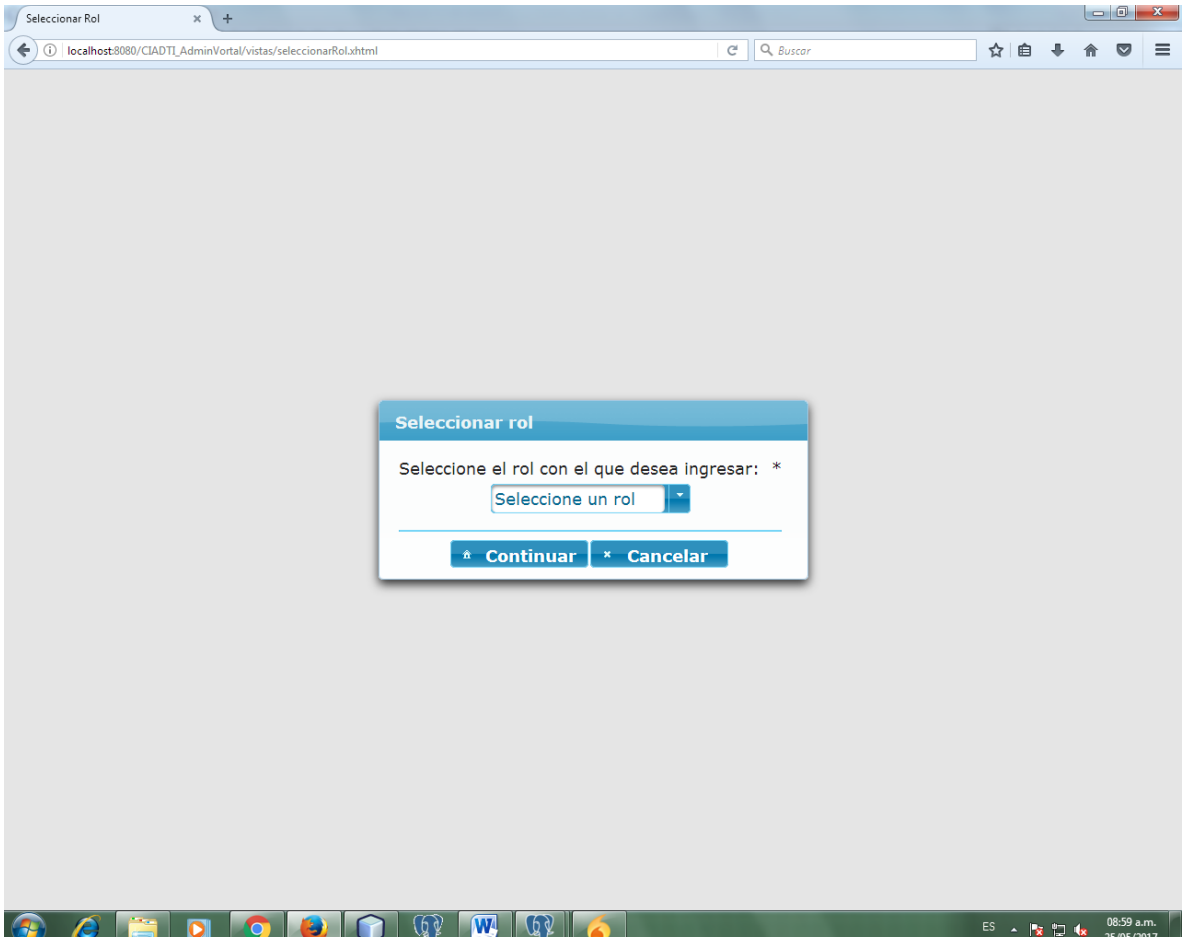


3.4.2.3 Roles de usuario

Después de loguearse correctamente el usuario es direccionado a la página seleccionarRol donde debe seleccionar uno de los roles según los que tenga asignados

para poder continuar (ver figura 53), este proceso es únicamente con el propósito de cargar las funcionalidades iniciales para el rol que ha sido seleccionado, el rol puede ser cambiado por el usuario después de estar adentro de la página administrador del Vortal.

Figura 53: Página de selección de rol



La página seleccionar rol consta de un cuadro de dialogo donde un elemento selectOneMenu es el encargado de mostrar los roles disponibles al usuario (ver figura 54), un botón continuar quien se encarga de validar la selección de alguno de los roles disponibles y direccionar hacia la página de inicio del administrador Vortal (ver figura 60) y un botón cancelar que se encarga de volver a la página de logueo y eliminar la sesión (ver figura 63).

Figura 54: Elemento selectOneMenu de la página seleccionarRol

```
<p:selectOneMenu id="selectOneMenuSeleccionarRol" style="width:155px"
                 required="true" requiredMessage="debe seleccionar un rol"
                 value="#{usuariorolControl.rolseleccionado}">
    <f:selectItem itemLabel="Seleccione un rol" itemValue=""
                  noSelectionOption="true" />

    <f:selectItems var="usurol"  value="#{usuariorolControl.rolesDelUsuario}"
                   itemLabel="#{usurol.rol.rolNombre}" itemValue="#{usurol.rol.rolNombre}"/>
    <p:ajax listener="#{usuariorolControl.actualizaRolSeleccionado}" event="change"/>
</p:selectOneMenu>
```

El atributo value del elemento selectOneMenu. Se encarga de actualizar el valor de la variable nombreRolSeleccionado en el Bean de respaldo usuariorolControl la cual almacena el nombre del rol que se selecciona.

Figura 55: Declaración de variable nombreRolSeleccionado en Bean usuariorolContol

```
private String nombreRolSeleccionado;

public String getNombreRolSeleccionado() {
    return nombreRolSeleccionado;
}

public void setNombreRolSeleccionado(String nombreRolSeleccionado) {
    this.nombreRolSeleccionado = nombreRolSeleccionado;
}
```

Cada vez que se selecciona un ítem diferente de la lista de roles, el elemento Ajax (ver figura 54) llama al método actualizaRolSeleccionado (ver figura 57) el cual cambia el valor del objeto rolSeleccionado (ver figuras 56).

Figura 56: Declaración de objeto rolSeleccionado en Bean usuariorolControl

```
private Rol rolSeleccionado;

public Rol getRolSeleccionado() {
    return rolSeleccionado;
}

public void setRolSeleccionado(Rol rolSeleccionado) {
    this.rolSeleccionado = rolSeleccionado;
}
```

Figura 57: Método actualizaRolSeleccionado en Bean usuariorolControl

```
public void actualizaRolSeleccionado() {
    if (nombreRolSeleccionado != null) {
        for (Usuariorol usurol : rolesDelUsuario) {
            if (usurol.getRol().getRolNombre().equals(nombreRolSeleccionado)) {
                FacesContext facesContex = FacesContext.getCurrentInstance();
                HttpSession sesion = (HttpSession) facesContex.getExternalContext().getSession(false);
                rolSeleccionado = usurol.getRol();
                sesion.setAttribute("rolSeleccionado", rolSeleccionado);
            } else {

            }
        }
    } else {
        System.out.print("rolseleccionado nulo ...");
    }
```

El elemento selectItems. Tiene los atributos value que hace referencia a la lista de roles disponibles para el usuario, la cual en el Bean de respaldo es representado por la variable rolesDelUsuario (ver figura 58); itemLabel que sirve para mostrar el nombre en pantalla por medio del objeto a medida que recorre la lista y el atributo itemValue para asignar un valor a cada ítem de la lista, de tal forma que se puede observar (ver figura 59).

Figura 58: Declaración de lista de objetos rolesDelUsuario en Bean usuariorolControl

```
private List<Usuariorol> rolesDelUsuario;

public List<Usuariorol> getRolesDelUsuario() {
    return rolesDelUsuario;
}

public void setRolesDelUsuario(List<Usuariorol> rolesDelUsuario) {
    this.rolesDelUsuario = rolesDelUsuario;
}
```

Figura 59: Aspecto visual del elemento selectItems de la página seleccionarRol

El botón continuar. Al presionarlo verifica si hay un elemento seleccionado y si es así pasa a la vista de la página de inicio, de lo contrario muestra el mensaje de validación en la parte superior del panel (ver figura 61).

Figura 60: El elemento commandButton para continuar en la página seleccionarRol

```
<p:commandButton value="Continuar"
                 icon="ui-icon-arrowthick-1-e ui-icon-home"
                 id="btnContinuarSesion"
                 style="width: 150px; height: 30px; text-align: center; font-weight: bold"
                 process="@this,selectOneMenuSeleccionarRol"
                 update=":formSeleccionarRol:msgs"
                 action="#{usuariorolControlador.cargarInicio()}"/>
```

Figura 61: Mensaje de error de validación para selección de rol

Figura 62: El elemento commandButton para cancelar en la página seleccionarRol

```
<p:commandButton value="Cancelar" icon="ui-icon-close"
                 id="btnCancelarAgregarUsuario"
                 style="width: 150px; height: 30px; text-align: center; font-weight: bold"
                 process="@this"
                 actionListener="#{loginControlador.cerrarSesion()}"/>
```

Cabe destacar que si el usuario únicamente tiene asignado un rol el sistema pasa de la página de logueo hacia la página de inicio directamente y carga las funcionalidades de ese rol ya que no tiene sentido entrar a la página de selección de rol si solo hay uno.

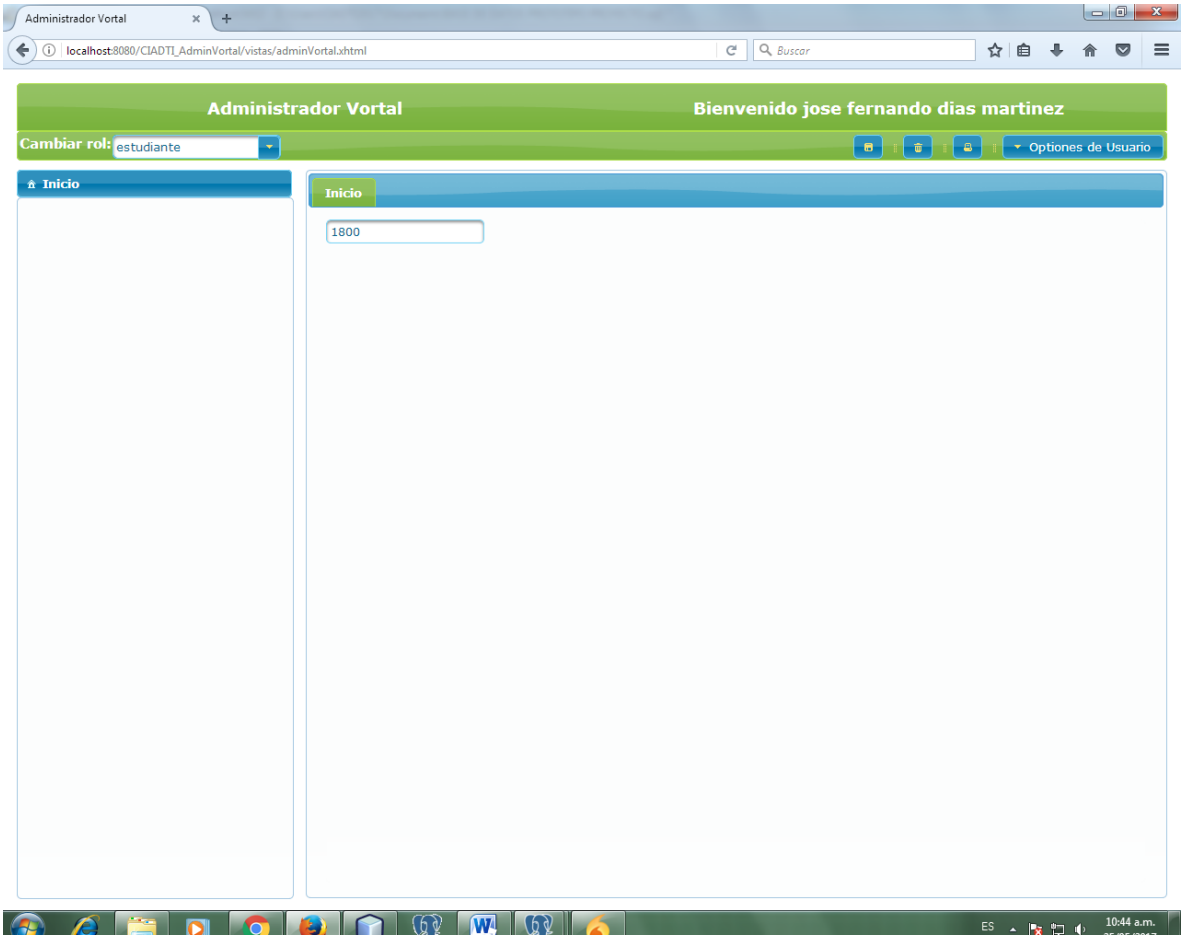
3.4.2.4 Página de inicio

Modelo de la página (Template). Es el contenedor de toda la página y está dividido por las secciones, cabecera, menú y contenido.

Figura 63: Modelo de la página de inicio

```
<div id="top" class="top">
    <ui:include src="cabecera.xhtml"/>
</div>
<div id="menu_contenido">
    <div id="left">
        <ui:include src="menuGeneral.xhtml"/>
    </div>
    <div id="content">
        <ui:include src="contenido.xhtml"/>
    </div>
</div>
```

Figura 64: Vista de página de inicio



La cabecera. Contiene el logo del administrador Vortal un mensaje de bienvenida para el usuario logueado y una barra que contiene un menú de selección para cambiar de

rol igual al visto en la página seleccionarRol (ver figura 59) y un menú de opciones de usuario donde se cierra la sesión (ver figura 64), el proceso de cierre de sesión se detallará más adelante en la sección 11.2.7.

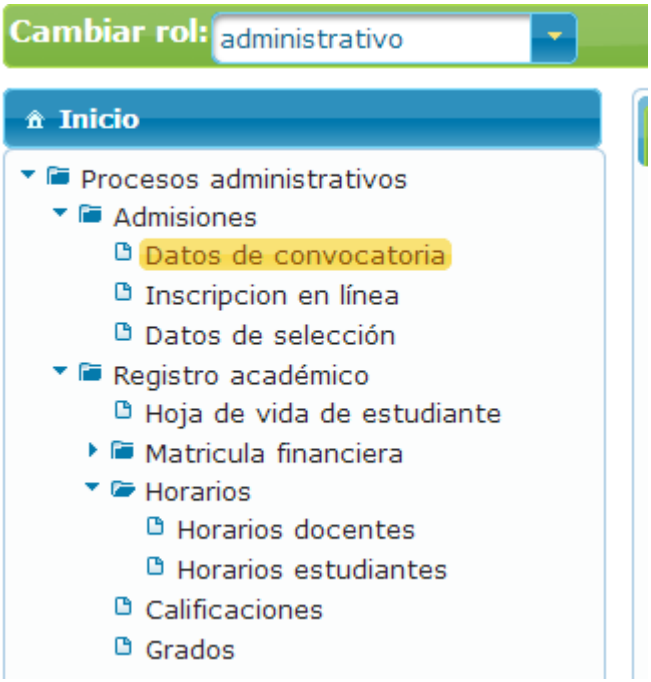
El menú. Contiene un árbol de funcionalidades, disponibles por cada rol seleccionado las funcionalidades pueden variar, este funcionamiento será detallado en la sección 11.2.5 y un botón inicio que carga el contenido inicial de la página.

El contenido. Es donde se abren las pestañas con el contenido necesario dependiendo de la funcionalidad seleccionada por el usuario, su funcionamiento será detallado en la sección 11.2.6.

3.4.2.5 Funcionalidades por rol

Las funcionalidades disponibles para el rol seleccionado se ven en pantalla en forma de árbol.

Figura 65: Vista de funcionalidades en forma de árbol



El código que hace posible la visualización de las funcionalidades tiene la siguiente estructura.

Figura 66: Código que pinta el árbol de funcionalidades en pantalla

```
<p:tree id="arbolFuncionalidades" value="#{rolfuncionalidadControl.root}" var="nodo" dynamic="true"
        style=" width: 99%; height: 760px; overflow:auto; display:block;"
        animate="true" selectionMode="single" cache="true"
        selection="#{rolfuncionalidadControl.nodoSeleccionado}">
    <p:treeNode icon="#{nodo.nombreIcono}" expandedIcon="ui-icon-folder-open"
                collapsedIcon="ui-icon-folder-collapsed">
        <p:outputLabel id="nombreNodoArbol" value="#{nodo.funcionalidad.funNombre}"/>
    </p:treeNode>
    <p:ajax  event="select"
             listener="#{tabsControl.agregarNewTab(nodo.funcionalidad,nodo.terminal)}"
             update=":formPestanas:tabViewFunc"/>
</p:tree>
```

El Bean de respaldo que permite llevar a cabo la lógica para la construcción del árbol se llama rolfuncionalidadControl y tiene una variable root de tipo TreeNode (ver figura 67) que es actualizada por el método construirArbol (ver figura 68) y este a su vez utiliza el método recursivo construyeHijos (ver figura 69).

Figura 67: Variable root del Bean de respaldo rolfuncionalidadControl

```
private TreeNode root;

public TreeNode getRoot() {
    return root;
}
```

Figura 68: Método construirArbol del Bean de respaldo rolfuncionalidadControl

```
public TreeNode construirArbol() {
    TreeNode raiz = new DefaultTreeNode("Root", null);
    for (Rolfuncionalidad rolFunc : listFuncionalidadesPorRol) {
        if (rolFunc.getFuncionalidad().getFunPadre() == null) {
            funcionalidadIcono FuncIcon = new funcionalidadIcono(
                                                    rolFunc.getFuncionalidad(),
                                                    "ui-icon-folder-collapsed", false
                                                    );
            TreeNode subNode = new DefaultTreeNode(FuncIcon, raiz);
            construyeHijos(subNode, rolFunc);
        }
    }
    return raiz;
}
```

Figura 69: Método construyeHijos del Bean de respaldo rolfuncionalidadControl

```
public void construyeHijos(TreeNode nodoPadre, Rolfuncionalidad FuncPadre) {
    boolean tieneHijo = false;
    for (Rolfuncionalidad rolFunc : listFuncionalidadesPorRol) {
        if (rolFunc.getFuncionalidad().getFunPadre() != null
                && rolFunc.getFuncionalidad().getFunPadre().equals(
                                                    FuncPadre.getFuncionalidad().getFunId())
                                                            ) {
            tieneHijo = true;
            funcionalidadIcono FuncIcon = new funcionalidadIcono(
                                                    rolFunc.getFuncionalidad(),
                                                    "ui-icon-folder-collapsed", false
                                                            );
            TreeNode subNode = new DefaultTreeNode(FuncIcon, nodoPadre);
            construyeHijos(subNode, rolFunc);
        }
    }
    if (!tieneHijo) {//si no tiene hijos le pone de icono document
        funcionalidadIcono FuncIcon = new funcionalidadIcono(
                                                    FuncPadre.getFuncionalidad(),
                                                    "ui-icon-document", true);
        TreeNode subNode = new DefaultTreeNode(FuncIcon, nodoPadre.getParent());
        nodoPadre.getParent().getChildren().remove(nodoPadre);
    }
}
```

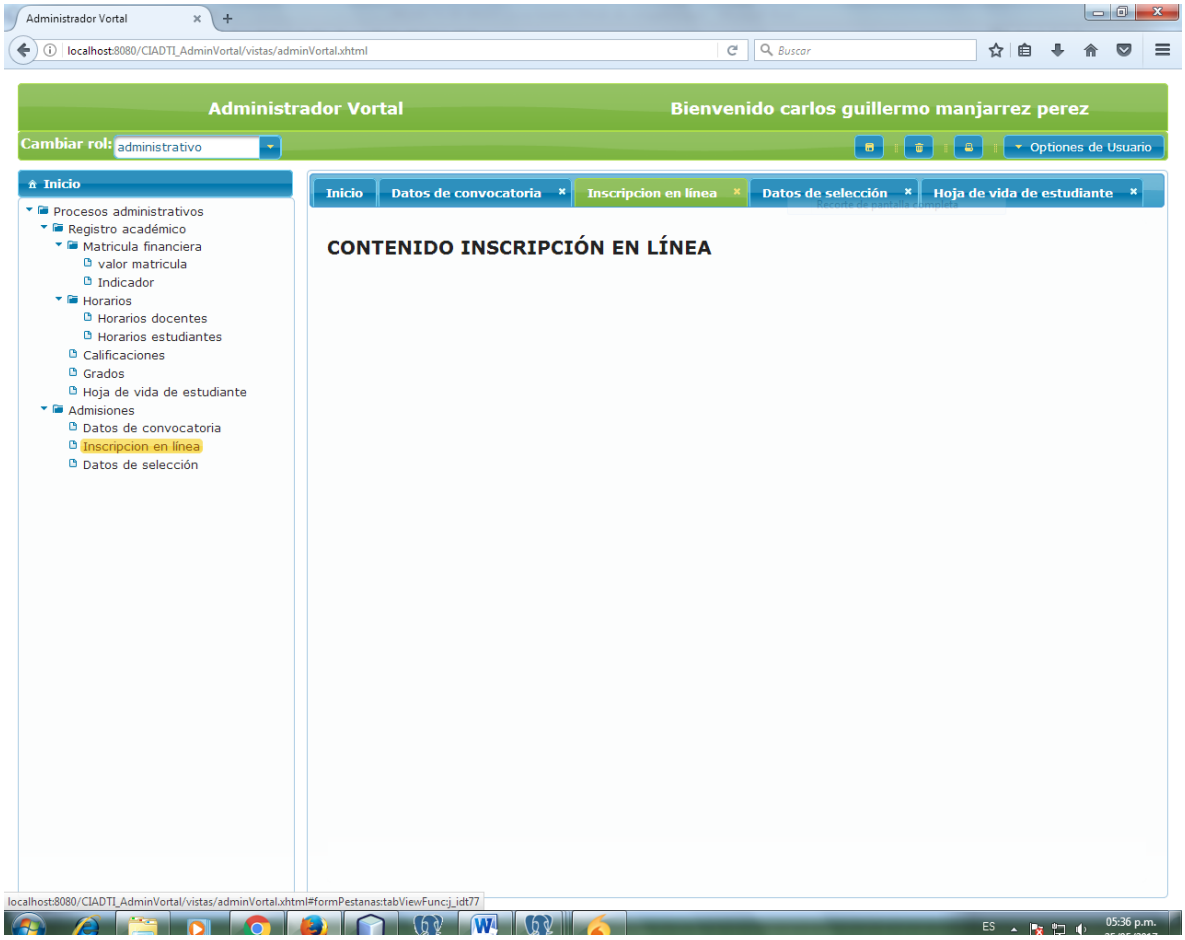
De esta manera el arbol se pinta en pantalla y al seleccionar una funcionalidad se abrirá una pestaña con su contenido correspondiente, ver la sección 11.2.6.

Al momento de construir el arbol de funcionalidades se tiene en cuenta un patrón caché que funciona almacenando los roles con sus respectivas funcionalidades para que cuando sea cargado el rol se pinten sus funcionalidades en menor tiempo y mejore la experiencia del usuario.

3.4.2.6 Manejo de contenido por pestañas

Cuando se selecciona una funcionalidad se construye una pestaña con su nombre y contenido donde se llevan a cabo los procesos correspondientes a dicha funcionalidad.

Figura 70: Vista de contenidos por pestañas



Cada vez que es seleccionada una funcionalidad terminal del árbol se ejecuta un método llamado agregarNewTab del Bean de respaldo llamado tabsControl.

Figura 71: Método agregarNewTab del Bean de respaldo tabsControl

```
public void agregarNewTab(Funcionalidad func, boolean nodoTerminal) {

    if (nodoTerminal) {
        boolean titRep = false;
        if (tabs.isEmpty()) {
            tabs.add(func);
            activeIndex = tabs.size();
        } else {
            for (Funcionalidad tabTemp : tabs) {
                if (tabTemp.getFunId().equals(func.getFunId())) {
                    titRep = true;
                    activeIndex = tabs.indexOf(tabTemp) + 1;
                }
            }
            if (!titRep) {
                tabs.add(func);
                activeIndex = tabs.size();
            } else {
                System.out.print("repetido...  " + func.getFunId());
            }
        }
    } else {
        System.out.print("no es nodo terminal  " + func.getFunId());
    }
}
```

El modelo encargado de pintar las pestañas en pantalla con sus respectivos nombres es el siguiente.

Figura 72: Código que permite visualizar las pestañas

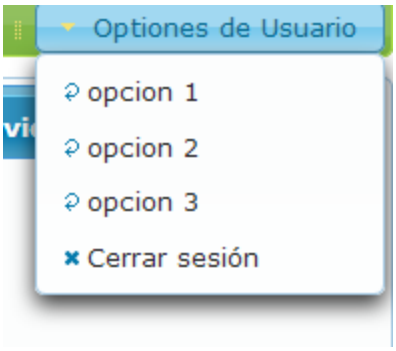
```
<p:tabView id="tabViewFunc" activeIndex="#{tabsControl.activeIndex}"
           cache="false" orientation="top">
    <p:tab title="Inicio" closable="false" id="inicio" >
        <p:scrollPanel style="width: 890px;height: 720px;border: none;"
                       mode="native" >
            <ui:include src="funcionalidad/contInicio.xhtml"/>
        </p:scrollPanel>
    </p:tab>
    <c:forEach items="#{tabsControl.tabs}" var="tab">
        <p:tab title="#{tab.funNombre}" closable="true">
            <ui:include src="/vistas/funcionalidad/#{tab.funUrl}.xhtml"/>
        </p:tab>
    </c:forEach>
    <p:ajax event="tabClose" listener="#{tabsControl.onTabCloseAction}"
            update=":formPestanas:tabViewFunc"/>
    <p:ajax event="tabChange" listener="#{tabsControl.onTabChangeAction}"/>
</p:tabView>
```

El elemento forEach itera sobre la lista de pestañas que se actualiza por medio de la variable tabs del Bean de respaldo tabsControl y pinta en pantalla cada tab con su respectivo título e incluye su contenido.

3.4.2.7 Cierre de sesión

En la parte superior derecha de la pantalla está disponible la opción de cierre de sesión para el usuario.

Figura 73: Opción de cierre de sesión en pantalla



El código que permite visualizar la opción de cierre de sesión es el siguiente.

Figura 74: código que permite visualizar la opción de cierre de sesión

```
<p:menuButton value="Optiones de Usuario" >
    <p:menuitem value="opcion 1" actionListener="#{toolbarView.save}" update="messages"
                icon="ui-icon-arrowrefresh-1-w" />
    <p:menuitem value="opcion 2" actionListener="#{toolbarView.update}" update="messages"
                icon="ui-icon-arrowrefresh-1-w" />
    <p:menuitem value="opcion 3" actionListener="#{toolbarView.delete}" ajax="false"
                icon="ui-icon-arrowrefresh-1-w" />
    <p:menuitem value="Cerrar sesión" action="#{loginControl.cerrarSesion()}"
                icon="ui-icon-close" />
</p:menuButton>
```

Al presionar la opción cerrar sesión el atributo action del subelemento menuitem contenido en el elemento menuButton invoca al método cerrarSesion del Bean de respaldo loginControl, quien se encarga de invalidar la sesión creada por el usuario.

Figura 75: Método cerrar sesión del Bean de respaldo loginControl

```
public void cerrarSesion() {
    try {
        ExternalContext externalContex = FacesContext.getCurrentInstance().getExternalContext();
        HttpSession sesion = (HttpSession) externalContex.getSession(false);
        externalContex.getSessionMap().clear();
        sesion.invalidate();
        externalContex.invalidateSession();
        externalContex.redirect(externalContex.getRequestContextPath() + "/loginVortal.xhtml");
    } catch (Exception e) {

    }
}
```

3.5 Análisis de los resultados

El resultados más importantes y que además era el propósito de este proyecto de investigación fue la selección del Framework con mayor calificación luego de hacer el estudio y análisis según los criterios planteados, este proceso arrojó como mejor opción al Framework web MVC open source JavaServer Faces con una puntuación significativa, lo que indica que al momento de hacer el desarrollo del prototipo bajo los beneficios que ofrece dicho Framework debían hacerse evidentes los potenciales que lo hicieron sobresalir por encima de los demás, y es

exactamente lo que sucedió, el Framework JSF mostró gran facilidad al momento de aprender a usar sus componentes; como por ejemplo; la integración e interacción entre los elementos de la vista y las variables y métodos de los Beans administrados; la integración entre elementos de Primefaces y variables y métodos de los Beans administrados; la integración de temas predefinidos de Primefaces y personalización de los mismos; la integración de clases Java para manejo de seguridad web (JAAS); la integración con APIs de persistencia de datos Hibernate, JPA, Ibatis, entre otros; la construcción de aspectos visuales con marcos dinámicos, llamados Templates; asignar ordenes directamente al servidor de aplicaciones por medio del archivo de configuración web.xml; asignar órdenes a su propio servlet para personalizar su comportamiento por medio del archivo faces-config.xml; inclusión de archivo para internacionalización de la aplicación; validación de elementos y campos en la vista por medio de atributos específicos; reconstrucción de elementos a partir de cambios en Beans administrados; integración con Ajax con el uso de tributos en botones y otros elementos para hacer llamado a métodos de Beans administrados; su arquitectura le permite construir la vista y entregar resultados a menor tiempo de ejecución; entre otras tareas que se pudo comprobar su facilidad sin perder la robustez de la aplicación.

4 Conclusiones

Se realizó la respectiva revisión bibliográfica con respecto a Frameworks web MVC open source en Java evidenciando y analizando sus características y arquitectura por medio de figuras ilustrativas tomadas de sus sitios oficiales o artículos en general disponibles en la web y se encontró qué, todos ofrecen beneficios para el desarrollador en cuanto a facilidad de uso, pero los más apropiados para este proyecto de grado son JSF y Spring.

Es cierto que hay variedad de Frameworks web MVC open source disponibles para construcción de aplicaciones en Java y se comprobó que no existe uno que se adapte a todo tipo de aplicativo, por ende la importancia de hacer un análisis para determinar el más apropiado según el tipo de aplicación que se pretende desarrollar y su alcance.

El proceso de validación de selección del Framework web MVC open source más apropiado se hizo basados en la metodología de selección de alternativas de solución del capítulo Administración de la Fundamentación del libro Ingeniería de Software Orientado a Objetos de Bernd Bruegge y Allen H. Dutoit.

Se desarrolló el prototipo de aplicación para el manejo de los procesos que actualmente lleva a cabo el aplicativo administrador Vortal en el CIADTI y se puso a prueba con algunas funcionalidades que pusieron a prueba su alcanzabilidad.

Con todo este proceso de investigación y aprendizaje la conclusión más significativa es el gran avance que las nuevas tecnologías han aportado al desarrollo de aplicaciones web con el fin de mejorar sus entornos y la experiencia en usuarios de los mismos.

5 Fuentes Bibliográficas

Anonimo. (2017). Modelo-Vista-Controlador. Retrieved from http://material.concursos.econo.unlp.edu.ar/concursos/Técnico-Profesional (Informática)/patrones/Modelo–vista–controlador.pdf

Bruegge, B., & Dutoit, A. H. (2002). *Ing. Soft. Orientado a Objetos* (primera ed).

Faraoni, F. (2014). DESARROLLO DE UNA APLICACIÓN WEB CON SPRING FRAMEWORK PARA UN GESTOR DE UN RECETARIO, 105. Retrieved from http://oa.upm.es/38731/1/TFG_Federico_Gutierrez_Faraoni.pdf

Fernández, Y., & Yanette, D. (2012). Telemática revista digital de las tecnologías de la información y las telecomunicaciones. Retrieved from http://webcache.googleusercontent.com/search?q=cache:http://revistatelematica.cujae.edu.cu/index.php/tele/article/view/15

GPSOS. (2017). Definición de Open Source - ¿Que es el Open Source? Retrieved May 26, 2017, from https://www.gpsos.es/soluciones-open-source/definicion-de-open-source/

Gutierrez, D. (2010). Frameworks y Componentes Universidad de los Andes. Retrieved from http://www.codecompiling.net/files/slides/IS_clase_10_frameworks_componentes.pdf

Hernán, V. (2013). ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN WEB QUE PERMITA AUTOMATIZAR LOS PROCESOS ADMINISTRATIVOS DEL CENTRO PSICOLOGICO DEL CAMPUS GIRÓN DE LA UNIVERSIDAD POLITÉCNICA SALESIANA, 150. Retrieved from http://dspace.ups.edu.ec/bitstream/123456789/5345/1/UPS-ST001028.pdf

JavaServer Faces. (2010). Capítulo 3. JavaServer Faces, 23. Retrieved from http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/viveros_s_ca/capitulo3.pdf

Javier, G. (2015). ¿Qué es un framework web?, 4. Retrieved from http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf

Microsoft. (2016). Model-View-Controller. Retrieved May 26, 2017, from https://msdn.microsoft.com/en-us/library/ff649643.aspx

OpenSource Initiative. (2007). The Open Source Definition | Open Source Initiative. Retrieved May 26, 2017, from https://opensource.org/osd

Pérez, J. (2013). Framework para la generación de aplicaciones orientadas al procesamiento de bio-señales, 113. Retrieved from http://www.bdigital.unal.edu.co/10588/1/71095012013.pdf

sicrea.com.mx. (2009). Struts2, 82. Retrieved from http://www.sicrea.com.mx/media/archivos/ManualStruts2Espanol.pdf

Sommerville, I. (2005). Ingeniería del Software. Retrieved from https://books.google.es/books?hl=es&lr=&id=gQWd49zSut4C&oi=fnd&pg=PA1&dq=inge

niería+de+software&ots=s678trszzi&sig=3upVrJnneR3hf3wQeWYbysgX5fE#v=onepage&q=ingeniería de software&f=false

Spring. (2012). Introduction to Spring Web MVC framework. Retrieved May 26, 2017, from https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html

Stallman, R. M. (2004). Software libre para una sociedad libre, 232. Retrieved from http://bibliotecadigital.org/bitstream/001/144/8/84-933555-1-8.pdf

Stallman Richard. (2016). gnu.org. Retrieved from http://www.gnu.org/philosophy/open-source-misses-the-point.html

TutorialsPoint. (2017). JSF Architecture. Retrieved May 26, 2017, from https://www.tutorialspoint.com/jsf/jsf_architecture.htm

UDLAP. (2013). Capítulo 3 : Spring, un framework de aplicación, 31. Retrieved from http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/sanchez_r_ma/capitulo3.pdf

Vaadin. (2011). *Vaadin 7 Edición*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.739.1565&rep=rep1&type=pdf

Vaadin. (2017). Vaadin Features. Retrieved May 26, 2017, from https://vaadin.com/features