ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE INFORMÁTICA Y ELECTRÓNICA

ESCUELA DE INGENIERÍA EN SISTEMAS

"ANÁLISIS COMPARATIVO DE LOS SERVIDORES GLASSFISH Y JBOSS PARA LA PLATAFORMA JAVAEE APLICADO AL MÓDULO DE CATÁLOGOS DEL SISTEMA DE RECURSOS HUMANOS DE LA ESPOCH"

TESIS DE GRADO PREVIA A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS

DIEGO PAUL TAMAYO BARRIGA

RIOBAMBA – ECUADOR

- 2014 -

AGRADECIMIENTO

Está tesis ha sido llevado a cabo para completar mis estudios de Ingeniería en Sistemas Informáticos de la Escuela Superior Politécnica de Chimborazo. Me gustaría dar las gracias a todas las personas que de algún modo me han ayudado a realizar este estudio, así como a todas las personas que me han educado como persona y me han formado académicamente. Especialmente quiero agradecer la educación y formación recibida por todos los maestros y docentes que me han impartido clases en el Instituto Tecnológico Superior “Juan de Velasco” y en la Escuela de Ingeniería en Sistemas Informáticos de la Escuela Superior Politécnica de Chimborazo.

De manera excepcional, quiero agradecer a Ivonne Rodríguez y Jorge Menéndez por su ayuda brindada en la dirección de la tesis. Me gustaría también agradecer a mis compañeros de mi primera relación laboral los cuales me ayudaron mucho en la formación para ser un gran profesional.

Con gran cariño, tengo que agradecer a mis amigos por los años compartidos durante la formación académica. Finalmente, tengo que dar mil gracias a Dios y a mis padres Angela y Octavio, por el apoyo y educación recibida constantemente. Creo y estoy muy orgulloso de decir que no hubiera podido vivir en ningún hogar mejor que en casa.

FIRMAS DE RESPONSABLES Y NOTAS

NOMBRES	FIRMA	FECHA
ING. IVÁN MENES CAMEJO **DECANO DE LA FACULTAD DE INFORMÁTICA Y ELECTRÓNICA**	_____	_____
ING. JORGE HUILCA **DIRECTOR DE LA ESCUELA DE INGENIERÍA EN SISTEMAS**	_____	_____
ING. IVONNE RODRÍGUEZ **DIRECTORA DE TESIS**	_____	_____
ING. JORGE MENÉNDEZ **MIEMBRO DE TESIS**	_____	_____
TLG. CARLOS RODRÍGUEZ **DIRECTOR DEL CENTRO DE DOCUMENTACIÓN**	_____	_____

NOTA: _____

RESPONSABILIDADES DEL AUTOR

Yo, Diego Paul Tamayo Barriga, soy el responsable de las ideas, doctrinas y resultados expuestos en esta tesis y el patrimonio intelectual de la misma que pertenece a la Escuela Superior Politécnica de Chimborazo.

_____

Diego Paul Tamayo Barriga

ÍNDICE DE ABREVIATURAS

AJAX	Asynchronous JavaScript And XML
AWT	Abstract Window Toolkit
API	Application Programming Interface
CPU	Central Processing Unit
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
JDBC	Java Database Connectivity
JDK	Java Development Kit
JMS	Java Message Service
JSF	JavaServer Faces
JSP	JavaServer Pages
JSR	Java Specification Request
JTA	Java Transaction API
PDA	Personal Digital Assistant
POJO	Plain Old Java Object
RAM	Random Access Memory
RMI	Java Remote Method Invocation
SSL	Secure Sockets Layer
URL	Uniform Resource Locator
XHTML	eXtensible HyperText Markup Language
XML	eXtensible Markup Language

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Actualmente en el campo de la Informática, se puede encontrar con avances tecnológicos diarios, el cual, proporciona y ayuda en la realización de nuestro trabajo diario.

Este hecho, hace que gran cantidad de sistemas informáticos estén desperdiciado muchos de sus recursos por una mala configuración realizados a estos o una mala elección del software. Además en el campo del desarrollo web también se ha avanzado notablemente gracias a las nuevas tecnologías que se han desarrollado por las diversas empresas tecnológicas que existen en la actualidad. Esto ha provocado que las nuevas plataformas sean muy eficientes. Pero de todas las ofertas de software de servidores disponibles en la actualidad.

Ante la aparición de las nuevas plataformas tecnológicas, existe una gran falta de información en la materia. Es todavía muy difícil encontrar documentos en la red que realicen un estudio comparativo de servidores confiables ya que las distintas empresas dan por ganador a su software sobre las demás.

Pero al no saber y contar con un estudio para la elección del mejor servidor la elección del software es por comodidad del Jefe del Proyecto o conocimientos previos de un desarrollador y no tienen un estudio de los servidores que compiten en el mercado. Esto provocaría al momento de su elección una reducción de costes económicos y un mejor aprovechamiento de los recursos hardware.

En esta línea surge la idea de realizar un estudio sobre una comparación de

los servidores libres para la plataforma Java, Enterprise Edition.

El objetivo principal de este trabajo es realizar una introducción a la plataforma Java, Enterprise Edition, presentar la herramienta de benchmarking sysstat y después realizar el estudio comparativo de los distintos servidores, para detectar cual es la mejor para cada ámbito y así justificar el porqué de la elección de ese servidor y no del otro.

Capítulo I

1. Marco de Referencia

1.1. Antecedentes

El avance informático a través del tiempo es cada vez más vertiginoso. Mientras ocurren los hechos, la tecnología, plataformas y los lenguajes de programación avanzan a velocidades que difícilmente se pueden estar al tanto de todo. El poder tener aplicaciones que puedan mantener procesos difícilmente llevados por otros medios y disponibles para todo el mundo, hace que sea necesario el desarrollo de estas.

Actualmente el Internet ha tomado fuerza en lo que se refiere al desarrollo de aplicaciones distribuidas.

En el Ecuador también se está automatizando la mayoría de instituciones por medio de sistemas web y con el Decreto 1014 se promueve el uso del Software Libre en las instituciones públicas, el cual establece su utilización en sus sistemas y equipos informáticos.

Para el desarrollo de una aplicación web ha surgido un numeroso conjunto de herramientas, según la empresa TIOBE Software que da a conocer el ranking mensual de la popularidad de lenguajes de programación, calculado a partir del número de resultados de los motores de búsqueda más populares en la actualidad, se puede apreciar que el lenguaje de programación Java es la herramienta más populares para el desarrollo web y además es una herramienta de software libre.

El Departamento de Sistemas y Telemática de la Escuela Superior Politécnica de Chimborazo siendo esta una institución pública necesita un sistema para el control del Recurso Humano y la coordinación de las distintas Unidades Académicas ya que en la actualidad los puestos de trabajo del Departamento de Recursos Humanos no cuentan con suficiente información, los procesos administrativos no se realiza de una forma ordenada y lógica, es decir no se realizan los procesos de descripción y valoración de los puestos de trabajo ya que no existe un normativo legal y aprobado para ello, esto conlleva a un desorden administrativo que da como resultado un déficit financiero, pobre distribución del personal y graves problemas en la gestión de Recursos Humanos.
El Departamento de Recursos Humanos mantiene la información del puesto de trabajo mediante una Hoja de Excel, el cual tiene como objetivo que las Unidades Académicas cuando necesite una vacante tenga la posibilidad de llenar esta ficha donde soliciten a Recursos Humanos, la contratación de un empleado para un puesto en particular.

Esto causa una falta de coordinación entre los distintos Departamentos ya que Recursos Humanos no tiene la suficiente información de la vacante

que se necesita, así como el Departamento Financiero no cuente con recursos para la nueva contratación.

Para resolver el problema planteado, se propone la creación de un sistema que permita minimizar el entorpecimiento de la contratación del personal. También se necesita implantar la aplicación web en un servidor de aplicaciones ya que gestiona la mayor parte de las funciones lógicas de negocio y acceso a datos de la aplicación.

Según la empresa Oracle Corporation propietaria de la plataforma Java muestra el listado compatible de servidores de aplicaciones para la plataforma Java EE 6, entre esta lista se tiene a Oracle GlassFish Server 3 y JBoss Application Server 7.1 como herramientas de software libre y en los que se puede implantar la aplicación a desarrollar.

Los principales beneficios de los servidores de aplicaciones son la centralización y la disminución de la complejidad en el desarrollo de aplicaciones.

Así los servidores de aplicación también incluye un middleware que permite la comunicación con varios servicios web, además nos brinda soporte a gran variedad de estándares como HTML, XML, JDBC, SSL, etc. que permite el funcionamiento en ambientes web de manera segura y la conexión a una gran variedad de fuentes de datos, sistemas y dispositivos.

Estos servidores de aplicaciones pueda ser considerado una solución para el despliegue de la aplicación web ya que cumplen con las especificaciones JSR 316: Java Plataform, Enterprise Edition 6 Specification.

Por eso la importancia de realizar un análisis comparativo de los servidores de aplicaciones para la plataforma Java EE 6, los cuales son herramientas de software libre de gran importancia en el mercado los cuales se adaptan y ajusta más a los requerimientos de la aplicación que se pretende desarrollar.

El presente trabajo tiene como objetivo conocer el servidor más eficiente en consumo de recursos para la plataforma Java EE 6 que permita optimizar recursos hardware del servidor, entonces:

Para poder decidir cuál es la mejor opción para la implantación de nuestra sistema es necesario conocer la cantidad de RAM que utiliza, la cantidad de disco duro que ocupa, tiempos de procesamiento por cada petición del usuario y tiempos que se demora el servidor en responder al usuario.

Para poder realizar el análisis comparativo se cuenta con diferentes herramientas para el análisis de los recursos que permite conocer el rendimiento y uso de recursos que ocupa los distintos servidores de aplicación de este análisis depende para la elección de cuál de estos servidores es el más óptimo para la implantación de la aplicación web.

1.2. Justificación

1.2.1. Justificación Teórica

Dentro de este contexto el presente análisis comparativo permite conocer el servidor más eficiente en consumo de recursos entre GlassFish y JBoss para la plataforma Java EE 6.

Dichos análisis se realizara por medio de herramientas benchmark el cual

indicara el comportamiento de cada uno los componentes hardware con el que cuenta el servidor. Esto permitirá conocer el uso de recursos y el rendimiento que tiene cada uno de los servidores de aplicación, de este análisis dependerá la elección del servidor óptimo para la implantación de la aplicación web.

1.2.2. Justificación Práctica

Para la comparación de los servidores de aplicaciones para la plataforma Java EE 6 se utilizara herramientas benchmark el cual es una técnica utilizada para medir el rendimiento de los componentes hardware del servidor.

Para el desarrollo del sistema se partirá del problema el cual es la necesidad de una organización en los diferentes departamentos para la contratación de un nuevo empleado para un puesto de trabajo, se propone la forma de hacer más organizado por medio de un sistema web el cual pueda ser fácil de utilizar así también que permita la coordinación entre los distintos Departamentos.

El desarrollo web del sistema se lo realizará con PrimeFaces un framework de JavaServer Faces que es parte de la plataforma Java EE 6, esta cuenta con un conjunto amplio de componentes que facilitan la creación de la aplicación y mejora la experiencia al usuario final.

El sistema se desarrollara con el fin de facilitar la cooperación entre los distinto departamento para la contratación de un nuevo empleado para un puesto de trabajo.

En esta investigación se desarrollara el módulo de catálogos para el

sistema de descripción y validación de puestos de trabajo del Departamento de Recursos Humanos, y este consta de los siguientes componentes que van hacer desarrolladas:

- Diseño Web
  - Maquetado de pantalla principal y pantallas secundarias
  - Creación de Hojas de Estilo para las distintas pantallas
  - Creación de plantillas JavaServer Faces para las distintas pantallas.
- Estructura de base de datos
  - Diseño del diagrama relacional
  - Creación de la base de datos
  - Creación de la capa de persistencia
- Desarrollo del módulo
  - Módulos de Catálogos

1.3. Objetivos

1.3.1. Objetivo General

Realizar el análisis comparativo de los servidores GlassFish y JBoss para la plataforma Java EE aplicado al módulo de catálogos del Sistema de Recursos Humanos de la ESPOCH.

1.3.2. Objetivos Específicos

- Estudiar los servidores de aplicaciones GlassFish y JBoss para la

plataforma Java EE 6.

- Determinar los criterios de análisis que permita determinar ventajas y desventajas para su posterior implantación del sistema web.
- Evaluar estadísticamente el rendimiento de los recursos y componentes hardware que consume cada servidor por medio de herramientas benchmark.
- Implementar el módulo de catálogos del sistema de descripción y validación de puestos de trabajo para el Departamento de Recursos Humanos, utilizando PrimeFaces que es un framework de JavaServer Faces que es parte de la plataforma Java EE 6.

1.4. Hipótesis

El servidor de aplicaciones GlassFish es más eficiente en el consumo de recursos en relación al servidor de aplicaciones JBoss.

Capítulo II

2. Marco Teórico

Actualmente en el campo de los computadores, se encuentra con avances en el hardware y software además gracias a esto proporciona una potencia de computo alto. Este hecho, hace que gran cantidad de herramientas se hayan creado con diversos propósitos.

Este capítulo se centra en la teoría del Lenguaje de Programación de Java especialmente en la Plataforma Java Enterprise Edition, en los servidores de Aplicaciones para Java, un benchmarking para comprobar el rendimiento de los servidores y una metodología de desarrollo.

En un análisis previo realizado en el Capitulo I se ha dado el porque de la elección del lenguaje de programación Java y de los servidores GlassFish y JBoss.

2.1. Plataforma Java

Es el nombre del entorno o plataforma de computación, capaz de ejecutar

aplicaciones desarrolladas usando el lenguaje de programación Java u otros lenguajes de programación y un conjunto de herramientas de desarrollo. La plataforma no es un hardware específico o un sistema operativo, sino más bien una máquina virtual encargada de la ejecución de las aplicaciones y un conjunto de bibliotecas estándar que ofrecen una funcionalidad común.

La plataforma Java puede ser utilizada desde móviles con hardware muy limitado hasta servidores web, entre las distintas plataformas que componen esta plataforma se tiene a la:

- *Plataforma Java, Edición Micro o Java ME.* Está plataforma permite crear aplicaciones para dispositivos con recursos limitados como PDA, móviles y algunos electrodomésticos.
- *Plataforma Java, Edición Estándar o Java SE.* Está plataforma permite crear aplicaciones para un entorno de escritorio.
- ***Plataforma Java, Edición Empresarial o Java EE.*** Está plataforma está pensado para el mundo web.

Los usuarios finales suelen interactuar con la máquina virtual de Java y un conjunto estándar de bibliotecas, así para un desarrollo de las aplicaciones, se utiliza un conjunto de herramientas que se les conoce como Java Development Toolkit o JDK.

Para la ejecución de un programa escrito en Java se necesita dos componentes:

1. **Máquina Virtual de Java:** el cuál es el núcleo de la plataforma es un concepto común de un procesador virtual que ejecuta programas escritos en el Lenguaje de Programación Java.

   En concreto ejecuta el código resultante de la compilación del código fuente, el cuál es conocido como bytecode. La máquina virtual es el encargado de traducir el bytecode en instrucciones nativas al lenguaje máquina de cada sistema operativo en el que se encuentre instalada.

   Esto permite que una misma aplicación Java pueda ser ejecutada en una gran variedad de sistemas operativos con arquitecturas distintas.

2. **Bibliotecas de Java:** Es el resultado de compilar el código fuente que incluye la Máquina Virtual de Java y ofrecer el apoyo para el desarrollo Java.

   - **Bibliotecas Centrales:** Funciones para la manipulación de distintos objetos como listas, archivos XML, etc.
   - **Bibliotecas de Integración:** que permite la comunicación con sistemas externos como acceso a base de datos, manejo de archivos.
   - **Bibliotecas para la Interfaz de Usuario:** para el manejo de interfaz de usuario para una mejor interacción como AWT, Swing.

El Lenguaje Java fue creado con cinco objetivos principales:

1. Debe usar la programación orientada a objetos.
2. Permitir la ejecución de una misma aplicación en múltiples sistemas operativos.

3. Incluir soporte para trabajo sobre la red.
4. Ejecutar código en sistemas remotos de forma segura.
5. Ser fácil de usar y tomar lo mejor de otros lenguajes de programación como Smalltalk o C++.

2.1.1. Historia

Java se creó como una herramienta de programación para ser usada en un proyecto de set-top-box en una pequeña operación denominada *The Green Project* en Sun Microsystems en el año de 1991. El equipo, compuesto por trece personas y dirigido por James Gosling, trabajó durante 18 meses en Sand Hill Road en Menlo Park en su desarrollo.

El lenguaje se denominó inicialmente *Oak*, luego pasó a denominarse *Green* tras descubrir que *Oak* era ya una marca comercial registrada para adoptadores de tarjetas gráficas y finalmente se nombró como *Java.*

Los objetivos de Gosling eran implementar una máquina virtual y un lenguaje con una estructura similar a C++. Entre junio y julio de 1994, tras una sesión maratónica de tres días entre John Gaga, James Gosling, Patrick Naughton, Wayne Rosing y Eric Schmidt, el equipo reorientó la plataforma hacia la Web. Sintieron que la llegada del navegador web Mosaic, propiciaría que Internet se convirtiera en un medio interactivo, como el que pensaba era la televisión por cable. Naughton creó entonces un prototipo de navegador, WebRunner, que más tarde sería conocido como HotJava.

En 1994, se les hizo una demostración de HotJava y la plataforma Java a

los ejecutivos de Sun. Java 1.0a pudo descargarse por primera vez en 1994, pero hubo que esperar al 23 de mayo de 1995, durante la conferencia de SunWorld, a que vieran la luz pública Java y HotJava. El acontecimiento fue anunciado por John Gage, el Director Científico de Sun Microsystems.

El acto estuvo acompañado por una pequeña sorpresa adicional, el anuncio por parte de Marc Andreessen, Vicepresidente Ejecutivo de Netscape, de que Java sería soportado en sus navegadores. El 9 de enero del año siguiente, 1996, Sun fundó el grupo empresarial JavaSoft para que se encargase del desarrollo tecnológico. Dos semanas más tarde la primera versión de Java fue publicada.

La promesa inicial de Gosling era *Write Once, Run Anywhere (Escríbase una vez, ejecútelo en cualquier lugar)*, proporcionando un lenguaje independiente de la plataforma y un entorno de ejecución ligero y gratuito para las plataformas más populares de forma que los binarios de las aplicaciones Java pudiesen ejecutarse en cualquier plataforma.

El entorno de ejecución era relativamente seguro y los principales navegadores web pronto incorporan la posibilidad de ejecutar Applets que es el lenguaje Java incrustado en las páginas web.

Java ha experimentado numerosos cambios desde la versión primigenia, JDK 1.0, así como un enorme incremento en el número de clases y paquetes que componen la biblioteca estándar.

Desde la versión 1.4 de Java, la evolución ha sido regulada por el Java Community Process para proponer y especificar cambios en la plataforma

Java. En abril del 2009 Oracle adquiere Sun Microsystems.

2.2. Plataforma Java, Edición Empresarial 6 o Java EE 6

Es una plataforma de programación para desarrollar y ejecutar software de aplicación. Permite utilizar arquitecturas de N capas distribuidas y se apoya ampliamente en componentes de software modulares ejecutándose sobre un mismo servidor de aplicaciones. La plataforma está definida por una especificación y dentro de esta cuenta con varias especificaciones que la componen como: JDBC, RMI, e-mail, JMS, Servicios Web, XML, etc. Java EE también configura algunas especificaciones únicas para Java EE para componentes, estas son EJB, Servlet, Portlets y varias tecnologías de servicio web. Ello permite al desarrollador crear una aplicación empresarial portable entre las diferentes plataformas.

Estos servidores incluyen un middleware que les permite intercomunicarse con varios servicios, para efecto de confiabilidad, seguridad, etc. Los servidores de aplicación también brindan a los desarrolladores una interfaz para programación de aplicaciones, de tal manera que no tenga que preocuparse por el sistema operativo o por la gran cantidad de interfaces requeridas en una aplicación web moderna.

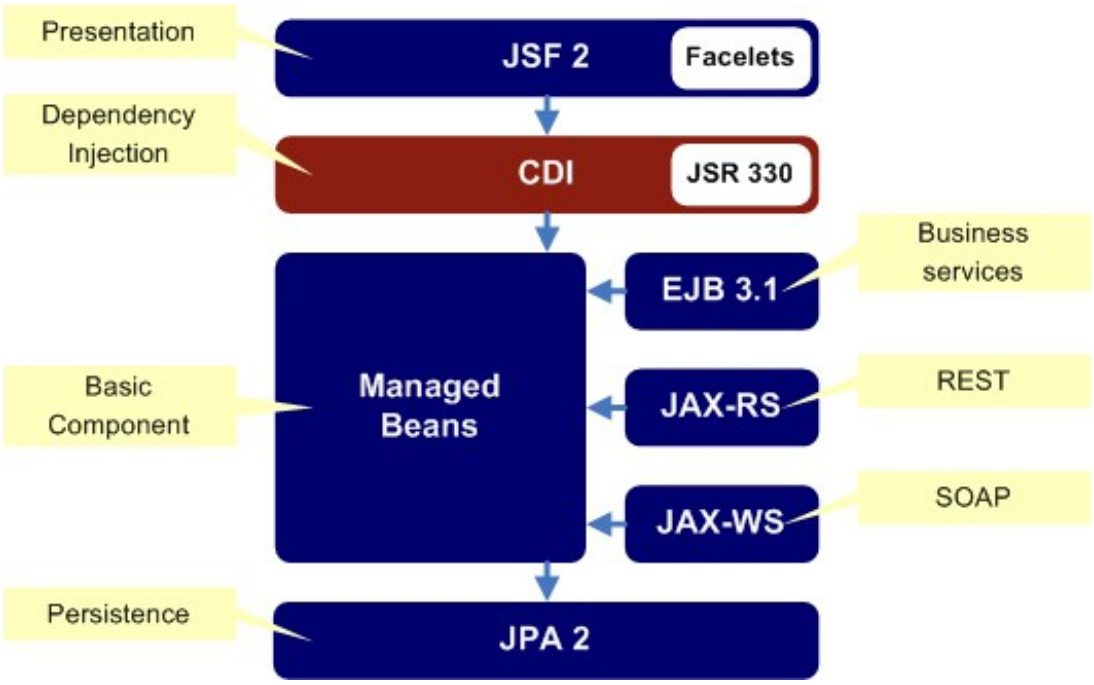


Figura 1: *Estándares que componen el framework de la Plataforma Java EE 6*

Fuente: http://www.youtube.com/watch?v=qhHmoVZCj8o&t=8m51s

2.2.1. Enterprise Beans[1]

Es un modelo de programación que permite construir aplicaciones Java mediante objetos simples (POJO). Cuando se construye una aplicación, son muchas las responsabilidades que se deben tener en cuenta, como la seguridad, transaccionabilidad, concurrencia, etc. El estándar EJB permite centrarse en el código de la lógica de negocio del problema que se desea solucionar y dejar el resto de responsabilidades al contenedor de aplicaciones donde se ejecutará la aplicación.

2.2.1.1. El contenedor de aplicaciones

Un contenedor de aplicaciones es un entorno que provee los servicios comunes a la aplicación que se desea ejecutar y gestionar. Dichos servicios

1 **JENDROCK E., Y OTROS.,** The Java EE 6 Tutorial., 2ª, ed., California - United States., 2013., Pg 433

incluyen la creación, mantenimiento y destrucción de objetos de negocio. Aunque el contenedor es responsable de la gestión y uso de dichos recursos y servicios, se puede interactuar con él para que nuestra aplicación haga uso de los servicios que ofrece.

Una vez escrita nuestra aplicación EJB, se puede desplegar en cualquier contenedor compatible con EJB, beneficiándose de todo el trabajo tras bastidores que gestionar eso en lugar del desarrollador.

De esta manera la lógica de negocio se mantiene independiente de otro código que pueda ser necesario, resultando en código que es más fácil de escribir y mantener.

2.2.1.2. La especificación EJB 3.1

Es parte de la plataforma Java EE 6 y este provee diversas APIs para la construcción de aplicaciones empresariales, entre ellas EJB, JPA, JMS y JAX-WS. Cada una de ellas se centra en un área específica, resolviendo problemas concretos. Además, cada especificación con su respectiva API está preparada para funcionar en compañía de las demás de forma nativa y por tanto en su conjunto son una solución perfectamente válida para desarrollar una aplicación end-to-end.

El uso de POJOs para encapsular la lógica de negocio proporciona un modelo simple que es altamente reutilizable. Se debe tener en cuenta que un POJO no actuará como un componente EJB hasta que haya sido empaquetado y desplegado en un contenedor EJB y accedido a dicho contenedor. Una vez que un POJO sea definido como EJB y este haya sido desplegado en el contenedor, se convertirá en uno de los tres siguientes

componentes:

1. Session Beans o Beans de Sesión
2. Message-Drive Bean
3. Entity Bean

2.2.1.3. Session Beans o Beans de Sesión

Son los componentes que contienen la lógica de negocio que requiere los clientes de la aplicación. Son accedidas a través de un proxy tras una solicitud al contenedor. Tras dicha solicitud, el cliente tiene una vista del Session Bean, pero no el Session Bean real. Esto permite al contenedor realizar ciertas operaciones sobre el Session Bean real de forma transparente para el cliente.

Los componentes Session Bean pueden ser de tres tipos:

1. ***Stateless Session Beans (SLSB):*** O Beans de Sesión Sin Estado, que son componentes que no requieren mantener un estado entre diferentes invocaciones. Un cliente debe asumir que diferentes solicitudes al contenedor de un mismo SLSB pueda devolver vistas a objetos diferentes. Dicho de otra manera, un SLSB puede ser compartido entre varios clientes. Por todo esto los SLBS son creados y destruidos a prudencia del contenedor y puesto que no mantiene el estado son muy eficientes a nivel de uso de memoria y recursos en el servidor.
2. ***Stateful Session Beans (SFSB):*** O Beans de Sesión Con Estado, sí que mantiene estado entre distintas invocaciones realizadas por

el mismo cliente. Esto permite crear un estado convencional, de manera que acciones llevadas a cabo en invocaciones anteriores son tenidas en cuenta para acciones posteriores. Un SFSB es creado justo antes de la primera invocación de un cliente, mantenido ligado a ese cliente y destruido cuando el cliente invoque un método en el SFSB que este marcado como finalizador (aunque también puede ser destruido por timeout de sesión). Son menos eficientes a nivel de uso de memoria y recursos en el servidor que los Beans de Sesión Sin Estado (SLSB)

3. ***Singleton:*** Es un nuevo tipo de Session Beans introducido en la especificación de EJB 3.1. Un Singleton es un componente que puede ser compartido por muchos clientes, de manera que una y solo una instancia es creada. A nivel de eficiencia en uso de recursos de memoria y recursos son indiscutiblemente los mejores, aunque su uso está restringido a resolver ciertos problemas muy específicos.

2.2.1.4. Message-Drive Beans o Beans Dirigido por Mensajes

Son componentes de tipo listener que actúan de forma asincrónica. Su misión es la de consumir mensajes, los cuales pueden gestionar directamente o enviar a otro componente. Los Message-Drive Bean actúan sobre un proveedor de mensajería por ejemplo Java Messaging System.

Al igual que los Stateless Session Beans, los Message-Drive Beans no mantienen estado entre invocaciones.

2.2.1.5. Entity Beans o Beans de Entidad

Son representaciones de datos almacenados en una base de datos, siempre en forma de POJOs. El encargado de gestionar los Entity Beans es `EntityManager`, un servicio que es suministrado por el contenedor y que está incluido en la especificación Java Persistence API. JPA es parte de EJB desde la versión 3.0. El tema de Interfaz de Persistencia Java (JPA)3 se presenta más adelante dentro de este informe.

2.2.2. Contextos e Inyección de Dependencia (CDI)[2]

Este es un patrón de diseño en el "que se suministra objetos a una clase en lugar de ser la propia clase quien los creen los objetos".

La forma habitual de implementar este patrón es mediante un contenedor de CDI y objetos simples en Java. El contenedor inyecta a cada objeto los objetos necesarios según las relaciones plasmadas en un fichero de configuración o anotaciones.

Típicamente este contenedor es implementado por un framework externo a la aplicación, por lo cual en la aplicación también se utilizará inversión de control al ser el contenedor que invoque el código de la aplicación.

Un claro ejemplo de uso de CDI se lo puede apreciar en uso de Managed Beans en JavaServer Faces8. Es una forma de inyectar un objeto a un Facelet y desde este poder usarlo.

2.2.3. Interfaz de Persistencia Java (JPA)[3]

JPA proporciona un modelo de persistencia basado en POJOs para

2 **JENDROCK E., Y OTROS.,** The Java EE 6 Tutorial., 2ª, ed., California - United States., 2013., Pg 513
3 **JENDROCK E., Y OTROS.,** The Java EE 6 Tutorial., 2ª, ed., California - United States., 2013., Pg 577

mapear bases de datos relacionales en Java. La persistencia de Java fue desarrollada por expertos de EJB 3.0 como parte de JSR 220, aunque su uso no se limita a los componentes software EJB. Se puede utilizar en aplicaciones web y aplicaciones clientes.

Para ello, combina ideas y conceptos de los principales frameworks de persistencia, como Hibernate, Toplink y JDO. El mapeo objeto-relacional (es decir, la relación entre entidades Java y tablas de la base de datos, query con nombre, etc) se realiza mediante anotaciones en las propias clases de entidad.

Pero para entender JPA, se tiene que tener en claro el concepto de "persistencia"

La persistencia o el almacenamiento permanente, es una de las necesidades básicas de cualquier sistema de información de cualquier tipo. En primer lugar, se propuso que el programa tratara los datos haciendo consultas directas a la base de datos. Después, se propuso trabajar con objetos, pero las bases de datos tradicionales no admiten esta opción.

Debido a esta situación, aparecieron los motores de persistencia, cuya función es traducir entre los dos formatos de datos: de registros a objetos y de objetos a registros. Persistir objetos Java en una base de datos relacional implica serializar un árbol de objetos Java en una base de datos de estructura tabular y viceversa. Esencial es la necesidad de mapear objetos Java para optimizar velocidad y eficiencia de la base de datos.

2.2.3.1. Unidad de persistencia

La unidad de persistencia define un conjunto de todas las entidades (clases) que son gestionadas por la instancia del `EntityManager` en una aplicación. Este conjunto de clases de entidades representa los datos contenidos en una única Base de Datos.

Las unidades de persistencia se definen en el fichero de configuración persistence.xml. Aquí se muestra un ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
             xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  <persistence-unit name="jobsPU" transaction-type="JTA">
    <jta-data-source>jdbc/jobs</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
    </properties>
  </persistence-unit>
</persistence>
```

Figura 2: *Archivo de configuración persistence.xml*

Realizado por: Diego P. Tamayo Barriga

La persistencia puede tener 4 estados diferentes:

- ***Transient:*** Un objeto recién creado que no ha sido enlazado con el gestor de persistencia.
- ***Persistent:*** Un objeto enlazado con la sesión (Todos los cambios serán persistentes).
- ***Detached:*** Un objeto persistente que sigue en memoria después de que termina la sesión: existe en Java y en la Base de Datos.

- ***Removed:*** Un objeto marcado para ser eliminado de la Base de Datos: existe en Java y se borrará de la Base de Datos al terminar la sesión.

2.2.3.2. EntityManager

Antes que nada se tiene que tener bien en claro dos temas muy importantes que son:

- Application-managed entity manager
- Container-managed entity manager

Para las situaciones en donde no requerimos los servicios ofrecidos por un EJB 3,0 Container, pero se quiere utilizar el modelo de persistencia de JPA, el API JPA provee el Application-managed entity manager. En estos casos las aplicaciones son denominadas aplicaciones standalone. Estas aplicaciones que corren fuera de un EJB 3,0 container (porque no lo necesitan) usaran el resource-local transaction provisto por el entity manager, lo que permite que la aplicación será la encargada de manejar el ciclo de vida del entity manager. Para las situaciones en donde sí se requiera de los servicios de un EJB 3,0 container (como por ejemplo JBoss o GlassFish), el API JPA provee el Container-managed entity manager.

Siempre que una transacción sea iniciada, un nuevo contexto de persistencia (persistence context) es creado. Esto es así tanto para el Application-managed entity manager como también para el Container-managed entity manager: Para el caso del Application-managed entity manager (cuando no se usa un application server), la aplicación es la encargada de abrir y cerrar la transacción.

Para el caso del Container managed entity manager (cuando se utiliza un EJB container), por defecto, la transacción es iniciada cuando se invoque desde el cliente al EJB (a un stateless session bean, ahora bien, para los stateful session bean el comportamiento de las transacciones es distinto). La transacción termina cuando finaliza la ejecución del método del session bean.

2.2.3.3. Interfaces JPA

Las interfaces que componen el JPA son:

`javax.persistence.Persistence:` Contiene métodos estáticos de ayuda para obtener una instancia de Entity Manager Factory de una forma independiente al vendedor de la implementación de JPA. Una clase de inicialización que va proporcionar un método estático para la creación de una Entity Manager Factory.

`javax.persistence.EntityManagerFactory:` La clase o entidad de `javax.persistence.Entity.Manager.Factory` ayuda a crear objetos de `EntityManager` utilizando el patrón de diseño Factory. Este objeto en tiempo de ejecución representa una unidad de persistencia particular. Generalmente va a ser manejado como un `singleton` y proporciona métodos para la creación de instancias `EntityManager`.

`javax.persistence.EntityManagerFactory:` La clase `javax.persistence.Entity` es una anotación Java que se coloca a nivel de clases Java serializables y que cada objeto de una de estas clases anotadas representa un registro de una base de datos.

javax.persistence.EntityManager: Es la interfaz principal de JPA utilizada para la persistencia de las aplicaciones. Cada Entity Manager puede realizar operaciones CRUD (Create, Read, Update, Delete) sobre un conjunto de objetos persistentes. Es un objeto único, no compartido que representa una unidad de trabajo particular para el acceso a datos. Proporciona métodos para gestionar el ciclo de vida de las instancias entidad y para crear instancias Query.

javax.persistence.Query: La interface `javax.persistence.Query` está implementada por cada vendedor de JPA para encontrar objetos persistentes manejando cierto criterio de búsqueda. JPA estandariza el soporte para consultas utilizando Java Persistence Query Language (JPQL) y Structured Query Language (SQL). Se puede obtener una instancia de **`Query`** desde una instancia de un `Entity Manager`.

javax.persistence.EntityTransaction: Cada instancia de Entity Manager tiene una relación de uno a uno, con una instancia de `javax.persistence.EntityTransaction`, permite operaciones sobre datos persistentes de manera que agrupados formen una unidad de trabajo transaccional, en el que todo el grupo sincroniza su estado de persistencia en la base de datos o todos fallan en el intento, en caso de fallo, la base de datos quedará con su estado original. Maneja el concepto de todos o ninguno para mantener la integridad de los datos.

2.2.3.4. Entidades

Una entidad es un objeto de dominio de persistencia. Normalmente, una entidad representa una tabla en el modelo de datos relacional y cada instancia de esta entidad corresponde a un registro en esa tabla.

El estado de persistencia de una entidad se representa a través de campos persistentes o propiedades persistentes. Estos campos o propiedades usan anotaciones para el mapeo de estos objetos en el modelo de base de datos.

El estado persistente de una entidad puede ser accesible a través de variables de instancia a la entidad o bien a través de las propiedades de estilo de Java Bean.

Las entidades podrán utilizar campos persistentes o propiedades. Si las anotaciones de mapeo se aplican a las instancias de las entidades, la entidad utiliza campos persistentes, En cambio, si se aplican a los métodos getters de la entidad, se utilizarán propiedades persistentes. Hay que tener en cuenta que no es posible aplicar anotaciones tanto a campos como a propiedades en una misma entidad.

Una entidad pasará a ser manejada por el contexto de persistencia de JPA cuando ésta sea persistida (mediante el método `persist()` del Entity Manager). En este punto, la entidad pasará a estar asociada a lo que comúnmente se llama al contexto de persistencia. En este caso, y mientras la entidad sea manejada/asociada por el contexto de persistencia (también se las conoce como entidades atachadas o attached entities), el estado (valores de la propiedades) de la entidad será automáticamente

sincronizado con la Base de Datos.

```
package jobs.model;
/**
 * Pojo para la tabla DEGREE
 * @author Diego Tamayo
 */
@Entity
@Table(catalog = "jobs", schema = "public", uniqueConstraints = {
    @UniqueConstraint(columnNames = {"name"})})
public class Degree implements Serializable, EntityDecorator {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(nullable = false)
    private Integer id;
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 500)
    @Column(nullable = false, length = 500)
      //Setter y Getter
}
```

Figura 3: *Ejemplo de una Entidad JPA*

Realizado por: Diego P. Tamayo Barriga

- ***Campos de persistencia permanente:*** Si la entidad utiliza campos persistencia permanente, los accesos se realizan en tiempo de ejecución. Aquellos campos que no tienen anotaciones del tipo `javax.persistence.Transient` o no han sido marcados como Java transitorio serán persistentes para el almacenamiento de datos. Las anotaciones de mapeo objeto/relación deben aplicarse a los atributos de la instancia.

- ***Propiedades de persistencia permanente:*** Si la entidad utiliza propiedades de persistencia permanente, la entidad debe seguir el método de los convenios de componentes Java Beans. Las propiedades de Java Bean usan métodos getters y setters en cuyo

nombre va incluido el atributo de la clase al cuál hacen referencia. Si el atributo es booleano podrá utilizarse `isProperty` en lugar de `getProperty`.

2.2.3.5. Relaciones Múltiples de la Entidad

Hay cuatro tipos de relaciones: uno a uno, uno a muchos, muchos a uno, y muchos a muchos.

1. ***Uno a uno:*** Cada entidad se relaciona con una sola instancia de otra entidad. Las relaciones uno a uno utilizan anotaciones de la persistencia de Java "`OneToOne`".
2. ***Uno a muchos:*** Una entidad, puede estar relacionada con varias instancias de otras entidades. Las relaciones uno a muchos utilizan anotaciones de la persistencia de Java "`OneToMany`" en los campos o propiedades persistentes.
3. ***Muchos a uno:*** Múltiples instancias de una entidad pueden estar relacionadas con una sola instancia de otra entidad. Esta multiplicidad es lo contrario a la relación uno a muchos. Las relaciones muchos a uno utilizan anotaciones de la persistencia de java "`ManyToOne`" en los campos o propiedades persistentes.
4. ***Muchos a muchos:*** En este caso varias instancias de una entidad pueden relacionarse con múltiples instancias de otras entidades. Este tipo de relación utiliza anotaciones de la persistencia de java "`ManyToMany`" en los campos o propiedades persistentes.

2.2.3.6. Transacciones

Representa un contexto de ejecución dentro del cual se puede realizar varias operaciones como si fuera una sola, de manera que todas ellas son, de manera que todas ellas son realizadas satisfactoriamente o el proceso se aborta en su totalidad. Esto es muy importante para garantizar la integridad de los datos que queremos persistir.

Si cualquiera de las entidades de esta lista no pudieran ser persistidas por algún motivo, no se desa que el resto de las entidades de la lista tampoco. Si permitimos que esto ocurriera, nuestros datos en la base de datos no reflejaría su estado real como objetos y ni datos. JPA permite configurar en cada unidad de persistencia el tipo de transacción que se quiere usar, que puede ser:

- Manejada por la aplicación (RESOURCE_LOCAL)
- Manejada por el contenedor (JTA)

Las transacciones deben cumplir cuatro propiedades:

1. ***Atomicidad:*** es la propiedad que asegura que la operación se ha realizado o no, y por lo tanto ante un fallo del sistema no puede quedar a medias.
2. ***Consistencia:*** es la propiedad que asegura que sólo se empieza y cuando se debe acabar. Por lo tanto, se ejecuta aquellas operaciones que no van a romper la reglas y directrices de integridad de la base de datos.
3. ***Aislamiento:*** es la propiedad que asegura que una operación no

puede afectar a otra. Esto asegura que la realización de dos transacciones sobre la misma información nunca generará ningún tipo de error.

4. ***Permanencia:*** es la propiedad que asegura que una vez realizada la operación, está persistirá y no se podrá deshacer aunque falle el sistema.

La atomicidad frente a fallos se suele implementar con mecanismos de journaling y la protección frente a accesos concurrentes mediante bloqueos de la estructura o tabla afectada. La permanencia se suele implementar forzando a los periféricos encargados de almacenar los cambios a confirmar la completa y definitiva transmisión de los datos al medio por lo general es a un disco.

2.2.4. Seguridad[4]

Para conseguir un buen sistema de seguridad se debe implementar buenas prácticas para conseguir un buen trabajo:

- ***No dar nunca nada por hecho:*** ni en cuestiones de seguridad ni en cuestiones del flujo normal de la aplicación. Todo el riesgo que se corra debe ser por parte del usuario (no hay nada que hacer contra eso). Explico, no se debe suponer que si le pido el nombre al usuario no me va a poner un número de teléfono. No sé si sería una forma acertada de decirlo pero hay que pensar con pesimismo, en los peores casos, y por remotos que sean pueden ocurrir.

4 **JAVA EE: SEGURIDAD EN APLICACIONES WEB**
http://jdiezfoto.es/informatica/java-ee-seguridad-en-aplicaciones-web-i/
2013/08/03

- ***Siempre que se use servicios externos se está asumiendo riesgos añadidos:*** Se puede haber hecho una página web muy segura y muy bien construida, pero si se incrusta contenido externo nadie puede asegurar que el contenido externo sea vulnerable a algún tipo de ataque.
- ***La oscuridad no es seguridad:*** No poner un botón de acceso a la administración no impide que se pueda acceder a ella. Ocultar nuestro código no debe ser parte de nuestra seguridad.
- ***Principio del mínimo privilegio:*** El usuario del sistema debe tener únicamente los privilegios que necesita para llevar a cabo su actividad.
- ***Fallar de manera segura:*** Hay que tratar de manera muy cuidadosa los fallos en la aplicación. Por poner un ejemplo, si se produce un fallo en la aplicación mientras se realiza tareas administrativas no debe seguir iniciada la sesión como administrador. Otro ejemplo, no debe mostrar en un fallo información técnica sobre el mismo al usuario del sistema. Si el usuario sabe datos acerca de nuestro sistema podría tener más fácil la búsqueda de vulnerabilidades.

2.2.4.1. Los riesgos

Ahora que se ha dado unos pequeños consejos sobre seguridad a la hora de la construcción de aplicaciones ya se puede pasar a explicar los riesgos que hay en las aplicaciones web.

Se listará los riesgos más importantes a continuación:

- ***Inyección SQL:*** Consiste en intentar “engañar” al sistema para que realice peticiones contra la base de datos que no son las que han sido programadas y que puede comprometer gravemente la base de datos o incluso mostrar al atacante toda la información de la misma.

- ***Cross Site Scripting:*** El atacante intentará enviar información al servidor por medio de formularios u otros medios con la intención de que dicha información sea almacenada en nuestra base de datos y posteriormente sea mostrada a los demás usuarios del sistema. Un ejemplo sencillo: Un código JavaScript que borre el contenido de la página, si eso es mostrado a los demás usuarios de la aplicación verán siempre una página en blanco. Esto es un ejemplo sencillo, pero imaginarios que lo que se consigue introducir es un código que tome el control de los navegadores de los usuarios de la aplicación web.

- ***Robo de sesión:*** Cómo se sabe HTTP es un protocolo sin estados, lo que significa que las credenciales o información de sesión deberá ir en cada petición; debido a esto dichos datos resultan muy expuestos. Un robo de estos datos podría tener como resultado que alguien se estuviera haciendo pasar por nosotros y realizando acciones con unos privilegios que se permita. Y tampoco se dé olvidar que se pude robar la sesión intentando obtener nuestras credenciales de alguna manera.

- ***Acceso a URLs restringidas:*** Consiste en la observación de una

URL e intentar cambiarla para intentar acceder a otras zonas. Estas es una de las razones por las que la seguridad a través de la ocultación no es efectiva.

2.2.4.2. Solucionando los problemas de seguridad

Ahora que ya se ha identificado y visto en qué consisten los problemas que se puede encontrar en las aplicaciones web, o al menos lo más importantes y peligrosos se va a ir uno por uno explicando cómo se puede ir solucionando. Primero a los dos últimos que son los más sencillos y en posteriores entradas se explicara la solución a los dos primeros que son los más importantes de los cuatro y quiero dedicarles una entrada completa.

- ***Robo de sesión:*** Ya se ha visto el riesgo que tiene el robo de una sesión. Se puede decir de manera resumida que el peligro está en la exposición de los datos de sesión. En la figura siguiente:

Figura 4: *Esquema de robo de sesión*

Fuente: http://jdiezfoto.es/wp-content/uploads/2011/10/RoboSesion.jpg

Para solucionar este problema de seguridad hay que atenerse a varios aspectos de la seguridad: la autenticación y la sesión; para cada uno de ellos se verá varios aspectos importantes a cubrir para solucionar problemas con el robo de sesión.

- ***La autenticación***

  - El más importante de todos. Usar SSL sobre HTTP (HTTPS) para transferir los datos y asegurarse de que el cifrado cubre los credenciales y el ID de sesión en todas las comunicaciones. De esta manera los datos de sesión de los que se hablaba siguen estando expuestos pero esta vez se encuentran cifrados, por lo que no se pueden usar. Alguien podría pensar: "¿Y si se obtiene la sesión y se rompe la encriptación?", la respuesta es sencilla, y es que con los medios actuales para cuando hayas conseguido romper la encriptación esa sesión habrá dejado de existir.

  - Usar un sistema de autentificación simple, centralizado y estandarizado. Es mejor que usar métodos de autentificación que se proporcione el propio servidor de aplicaciones en vez de soluciones implementadas por el desarrollador, debido a que lo que implementa el servidor de aplicaciones es usado en muchos lugares y ha sido suficientemente probado. Por ejemplo los filtros de Java EE o los métodos de autentificación que proporciona Java EE.

  - Posibilitar el bloqueo de autentificación después de un

número determinado de intentos fallidos. Esto podría evitar ataques de fuerza bruta intentando averiguar la contraseña del usuario.

- Implementar métodos seguros de recuperación de contraseñas: Es común que se intente usar estos métodos para intentar ganar acceso a una cuenta del usuario, se puede ver una serie de consejos para implementar estos métodos. Pedir al usuario al menos tres datos o más, obligar a que responda preguntas de seguridad. La contraseña que es recuperada deberá generarse de manera aleatoria y enviada al usuario por un canal diferente, de esta forma si el atacante consiguió sortear los primeros pasos es difícil que logre sortear el canal usado para transmitir.

- ***La sesión***

  - Usar los métodos de sesión que se proporciona el servidor de aplicaciones que se esté usando, y digo esto por las mismas razones que aconsejé usar los métodos de autentificación que proporciona el servidor de aplicaciones. En este caso no está refiriendo a la sesión y a las cookies.

  - Asegurar que la operación de cierre de sesión realmente destruye dicha sesión. También fijar el periodo de expiración de la sesión, por ejemplo para aplicaciones críticas de 2 a 5 minutos, mientras que para otras aplicaciones más comunes se podría usar de 15 a 30 minutos.

En el descriptor de despliegue se puede fijar la caducidad de la sesión en minutos dentro del framework Java EE.

```
<session-config>
      <session-timeout>15</session-timeout>
</session-config>
```

Figura 5: *Configurando el tiempo de sesión en el archivo web.xml*

Realizado por: Diego P. Tamayo Barriga

- ***Acceso a URLs restringuidas:*** Como se ha visto en el aparato teórico sobre este problema de seguridad a traves de la ocultación no sirve de nada. Si por ejemplo para la parte pública se usará un patrón de URL, el atacante podría pensar que la parte privada pudiera ser /admin o cosas parecidas, con no dar a conocer el detalle de este directorio no es suficiente, hay que protegerlo.

  Como a estas alturas se supone no hay que hacer nada especial si se ha realizado lo anterior, es decir, si se ha realizado una autenticación y un control de sesión correctos. Con el uso de los filtros se puede solucionar este problema perfectamente.

  Sé que es un problema que a la vista parece bastante evidente y con una solución sencilla; pero mientras siga apareciendo como uno de los problemas más graves de seguridad será porque no está tan bien solucionado aunque sea evidente el problema y la solución.

2.3. JavaServer Faces[5]

Es una tecnología y framework para aplicaciones Java basadas en la web, que simplifica el desarrollo de interfaces de usuarios en aplicaciones Java EE. JSF usa tecnologías como XUL (acrónimo de XML-based User-Interface Language, Lenguaje basado en XML para interfaz de usuario).

JSF dentro de su especificación incluye:

- Un conjunto de APIs para representar componentes de una interfaz de usuario y administrar su estado, manejando eventos, validación de entradas, definir un esquema de navegación de las páginas y dar soporte para internacionalización y accesibilidad.
- Un conjunto por defecto de componentes de interfaz de usuario.
- Dos bibliotecas de etiqueta personalizadas para XUL que permite expresar una interfaz de usuario.
- Un modelo de eventos en el lado del servidor.
- Administración de estados.
- Beans administrados.

2.3.1. Beneficios de la tecnología JavaServer Faces

Las principales ventajas de JavaServer Faces son:

- ***Componentes personalizados:*** JavaServer Faces permite combinar fácilmente complejas GUIs en un componente.
- ***Mejor soporte para Ajax:*** Una gran parte de librerías de componentes dan soporte Ajax.

5 **JENDROCK E., Y OTROS.,** The Java EE 6 Tutorial., 2ª, ed., California - United States., 2013., Pg 103

- ***Ofrecer soporte para otras tecnologías display:*** JavaServer Faces no está limitado a HTML y HTTP.
- ***Acceso a beans por nombre:*** JavaServer Faces te permite asignar nombres a beans, con lo que se puede referenciar en los formularios.
- ***Lenguaje de expresión:*** El lenguaje de expresión de JavaServer Faces es más conciso y potente.
- ***Controladores y definiciones de beans más simples***: JavaServer Faces no exige que tu controlador y las clases beans sean extendidas a algunas clases padre en particular o usen algún método en particular.
- ***Herramientas más potentes:*** JavaServer Faces ofrece controles GUI y manejadores que facilitan el uso de IDEs de arrastrar y soltar.

2.3.2. Introducción de Facelets[6]

Es un framework para plantillas (templates) centrado en la tecnología JSF (JavaServer Faces), por lo cual se integran de manera muy fácil. Este framework incluye muchas características siendo las más importantes:

- Tiempo de desarrollo cero de los tags para UIComponents.
- Facilidad en la creación del templating para los componentes y páginas.
- Habilidad de separar los UIComponents en diferentes archivos.
- Un buen sistema de reporte de errores.

6 **JENDROCK E., Y OTROS.,** The Java EE 6 Tutorial., 2ª, ed., California - United States., 2013., Pg 111

- Soporte completo a EL (Expression Language).
- Validación de EL en tiempo de construcción.
- No es necesaria configuración XML.
- Trabaja con cualquier RenderKit.

Desafortunadamente JSP (JavaServer Pages) y JSF no se complementan naturalmente, cuando se usan juntos ambos escriben output al response, pero lo hacen de una manera diferente: JSP crea output ni bien encuentra código JSP (es decir procesa los elementos de la página de arriba a abajo), mientras que JSF dicta su propio re-rendering (ya que su ciclo de vida está dividido en fases marcadas). Facelets llena este vacío entre JSP y JSF, siendo una tecnología centrada en crear árboles de componentes y estar relacionado con el complejo ciclo de vida JSF.

2.3.2.1. ¿Por qué Facelets?

- No depende de un contenedor Web.
- Integrar JSP con JSF trae problemas, además, no se puede usar JSTL (JavaServer Pages Standard Tag Library) con JSF, cosa que Facelets sí provee.
- Facelets provee un proceso de compilación más rápido que JSP.
- Provee templating, lo cual implica reutilización de código, simplificación de desarrollo y facilidad en el mantenimiento de grandes aplicaciones.
- Permite crear componentes ligeros sin necesidad de crear los tags de los UIComponents (es más fácil comparado a crear un

componente JSF puro).

- Soporta Expression Language, incluyendo soporte para funciones EL y validación de EL en tiempo de compilación.

2.3.3. Expression Language[7]

El lenguaje EL utilizado en JavaServer Faces está especialmente diseñado para soportar el sofisticado modelo de componentes de interfaz de usuario, que permite realizar validaciones y conversiones, propagar datos de los componentes a los objetos y recoger los eventos de los componentes. Para ello, este lenguaje ofrece las siguientes funcionalidades:

- Evaluación diferida de expresiones
- Invocación de métodos
- Recoger y asignar datos

En el lenguaje EL hay dos formas de evaluar una expresión: de forma inmediata o diferida. La forma inmediata es la que se utiliza por ejemplo en la tecnología de páginas JavaServer Pages y la diferida se utiliza en las páginas JavaServer Faces.

La forma inmediata significa que la expresión será evaluada inmediatamente, convertida y rápidamente enviada a la etiqueta en forma de valor (estas expresiones son sólo de lectura).

En caso de las expresiones diferidas, serán tratadas en las diferentes fases del ciclo de vida y dependiendo de cada fase su tratamiento será diferente (puede ser de lectura o escritura). A continuación se muestra un

7 **JENDROCK E., Y OTROS.,** The Java EE 6 Tutorial., 2ª, ed., California - United States., 2013., Pg 125

ejemplo en el que se utiliza el lenguaje para dar el valor correspondiente en JavaServer Faces a un input de texto.

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

Figura 6: *lenguaje de Expresión diferida*
Realizado por: *Diego P. Tamayo Barriga*

En el lenguaje unificado EL se definen dos tipos de expresiones: expresiones de valor y expresiones de método.

2.3.3.1. Expresiones de Valor

Este tipo de expresiones pueden acceder a un valor o asignarlo. Existen dos tipos: RValue aquellas en las que el valor es de solo lectura y se utilizan en la evaluación inmediata o LValue en las cuales los datos pueden ser leídos y escritos, y son utilizados en la evaluación diferida.

Las primeras utilizan los delimitadores `${}` y las segundas `#{}`.

2.3.3.2. Expresiones de método

Son utilizadas para la llamada diferida a un método, que puede devolver un valor. En JavaServer Faces, estas expresiones son utilizadas para llamar a métodos que realizan algún proceso asociado al componente. Por ejemplo, estos métodos son necesarios para la gestión de eventos y para las validaciones. A continuación se muestra un ejemplo en el que mediante la llamada a un método se valida el nombre de un cliente y más tarde, un botón produce un evento recogido por otro método asociado a éste:

```
<h:form>
   <h:inputText id="name" value="#{cliente.nombre}"
      validator="#{cliente.validarNombre}"/>
   <h:commandButton id="enviar" action="#{cliente.enviarNombre}" />
</h:form>
```

Figura 7: *Ejemplo de expresión de método*

Realizado por: Diego P. Tamayo Barriga

Las expresiones de método sólo pueden ser usadas como atributos de las etiquetas:

- Con una construcción simple donde la clase será un componente Java Bean y el método de dicho componente
- Con un literal, solo texto, que actuó como el retorno del método

2.3.3.3. Análisis de expresiones

El lenguaje unificado EL dispone de APIs para realizar el análisis y la resolución de las distintas expresiones. Las principales clases del API son:

- *ValueExpression* que define una expresión de valor.
- *MethodExpression* que define una expresión de método.

Además, otras características que destacan de esta API son:

- El conjunto de implementaciones para resolver el lenguaje, en el que cada una se corresponde con un tipo de objetos o propiedad en particular.
- El objeto *ELContext* que almacena el estado de las resoluciones del lenguaje EL y almacena otros objetos.

El programador puede crearse sus propias clases para la resolución de las expresiones. En este lenguaje las expresiones de valor hacen uso de los

métodos *getValue* de la propiedad del objeto en cuestión para conseguir los valores a representar en las expresiones de tipo *RValue* en las de tipo *LValue* hacen también uso del método *setValue* para asignar valores.

2.3.3.4. Operadores EL

Los siguientes operadores pueden ser utilizados únicamente en el caso de las expresiones de valor:

Tabla I: *Operadores EL*

Operadores	
Aritmética	`+, -, *, /, div, %, mod, - (unario)`
Lógicos	`and, &&, or, ||, not, !`
Relacionales	`==, eq, !=, ne, <, lt, >, gt, <=, ge, >=, le`
Empty	`para determinar si un valor es nulo o vacío.`
Condicionales	`A ? B : C. Ejecuta o evalúa B o C, dependiendo de si A es cierto o falso`

La preferencia de los operadores de mayor a menor, y de izquierda a derecha es:

Tabla II: *Preferencias de operadores EL*

Preferencia de Operadores
• `[]` • `()` • `- (unitario) not! empty` • `/ div % mod` • `+ -` • `< > <= >= lt gt le ge` • `== != eq ne` • `&& and`

- || or
- ? :

2.3.4. Managed Beans en JavaServer Faces[8]

JavaServer Faces separa la capa de presentación de la lógica de negocio. Para que las páginas JSF puedan acceder a esta lógica se utiliza Managed Beans.

Un Bean es un POJO, algo tan sencillo como una clase que tiene un constructor público sin argumentos y sus propiedades tienen asociado sus correspondientes get/set.

No se los necesita instanciar, ya que son inicializados por su contenedor en tiempo de ejecución cuando la aplicación los necesite.

2.3.4.1. ¿Cómo declarar un Managed Beans?

Se puede declarar de dos maneras, por medio de un archivo de configuración o por medio de anotaciones dentro del POJO.

Para mayor eficiencia se lo debe realizar por medio de anotaciones, este proporciona un mejor control del código.

8 **JENDROCK E., Y OTROS.,** The Java EE 6 Tutorial., 2ª, ed., California - United States., 2013., Pg 189

1. Por medio de un archivo de configuración, faces-config.xml

```
<managed-bean>
   <managed-bean-name>userBean</managed-bean-name>
   <managed-bean-class>com.examples.UserBean</managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Figura 8: *Declaración de un Managed Beans utilizando, faces-config.xml*

Realizado por: Diego P. Tamayo Barriga

2. Por medio de anotaciones dentro del POJO.

```
package com.examples;
import javax.faces.bean.ManagedBean
import javax.faces.bean.SessionScoped

@ManagedBean
@SessionScoped
public class UserBean {...}
```

Figura 9: *Declaración de un Managed Beans utilizando anotaciones*

Realizado por: Diego P. Tamayo Barriga

2.3.4.2. Alcance de un Managed Beans dentro de una aplicación

El alcance no es más que un mapeo entre nombres y objetos que se almacena durante un determinado tiempo. Entre los alcances que vienen incluido son:

1. ***Application:*** Se guarda la información durante toda la vida de la aplicación, independiente de todas las peticiones y sesiones que se realice.

   Este bean se instancia con la primera petición y desaparece cuando la aplicación web se elimina del servidor.

2. ***Session:*** Cómo su nombre lo indica, este alcance guarda la información desde que el usuario comienza una sesión hasta que está termina.

3. ***View:*** Este alcance dura desde que se muestra una página JSF al usuario hasta que el usuario navega hacia otra página. Es muy útil para páginas que usan AJAX.

4. ***Request:*** Comienza cuando se envía una petición al servidor y termina cuando se devuelve la respuesta al usuario.

5. ***None***: Los beans se instancian cuando se necesite por otros beans, y se elimina cuando esta relación desaparece.

Cómo *buenas prácticas,* se recomienda utilizar siempre el menor alcance, para evitar así problemas de memoria al tener que almacenar más información de la necesaria.

2.3.5. El ciclo de vida de JavaServer Faces[9]

El ciclo de vida de un JavaServer Faces comienza cuando un usuario hace una petición por medio de un navegador por medio del protocolo HTTP/HTTPS al servidor y termina cuando el servidor responde la página correspondiente.

Cómo HTTP/HTTPS es un protocolo sin estado, no es capaz de recordar las transacciones anteriores que se han llevado a cabo entre el usuario y el servidor.

9 **JENDROCK E., Y OTROS.,** The Java EE 6 Tutorial., 2ª, ed., California - United States., 2013., Pg 210

2.3.5.1. Escenarios

JavaServer Faces admiten dos tipos de peticiones y dos tipos de respuestas:

- Respuesta Faces: La respuesta es generada en la fase de renderizado
- Respuesta No-faces: La respuesta no es generada en la fase de renderizado. Por ejemplo, un fichero JavaServer Page que no incluye componentes JSF
- Petición Faces: La petición es enviada desde una respuesta anterior generada por JSF. Por ejemplo, el envío de un formulario creado usando componentes JSF
- Petición No-faces: Se realiza una petición que es enviada a un componente como un servlet o un JavaServer Pages, en lugar de a un componente JSF

Combinando estas posibilidades, se obtienen tres escenarios posibles para el ciclo de vida de una aplicación JavaServer Faces - lógicamente, la combinación de petición y respuesta no-faces no involucra en ningún momento a las JavaServer Faces.

1. ***Escenario 1: Una petición no-faces genera una repuesta faces***

   Es el caso típico de un enlace en una página HTML que enlaza a una página JSF. En este caso, se debe configurar una correspondencia entre URL y `FacesServlet`. Cuando se genera la respuesta, la aplicación debe crear una nueva vista, almacenarla

en el contexto, obtener todas las referencias que necesite y renderizar inmediatamente la página.

2. ***Escenario 2: Una petición faces genera una repuesta no-faces***

   En este caso una aplicación JavaServer Faces necesitaría redirigir a un sitio diferente, o generar una respuesta que no contiene componentes JavaServer Faces.

   El desarrollador debe especificar que se omita la fase de renderizado - `FacesContext.responseComplete` - , lo que puede hacerse en las fases:

   - Aplicación de los valores de la petición
   - Validación
   - Actualización de modelo

3. ***Escenario 3: Una petición faces genera una repuesta faces***

   Este es el escenario más común: un componente JavaServer Faces envía una petición a una aplicación JavaServer Faces empleando el `FacesServlet`. Como es la implementación de JavaServer Faces quien toma el control, no es necesario hacer nada especial para generar la respuesta. Todos los eventos, validadores y convertidores se llamarán durante la fase apropiada del ciclo de vida estándar

2.3.5.2. Ciclo de Vida Estándar

JavaServer Faces soluciona esta falta de memoria manteniendo vistas en el lado del servidor. Una vista es un árbol de componentes que representa

la UI del usuario. Así, mientras que nos centramos en desarrollar los componentes, el ciclo de vida de un JavaServer Faces se preocupa de sincronizar estas vistas del lado del servidor y lo que se muestra al usuario.

Cuando un usuario pincha en un botón o un enlace comienza el ciclo de vida JSF que contiene las siguientes fases:

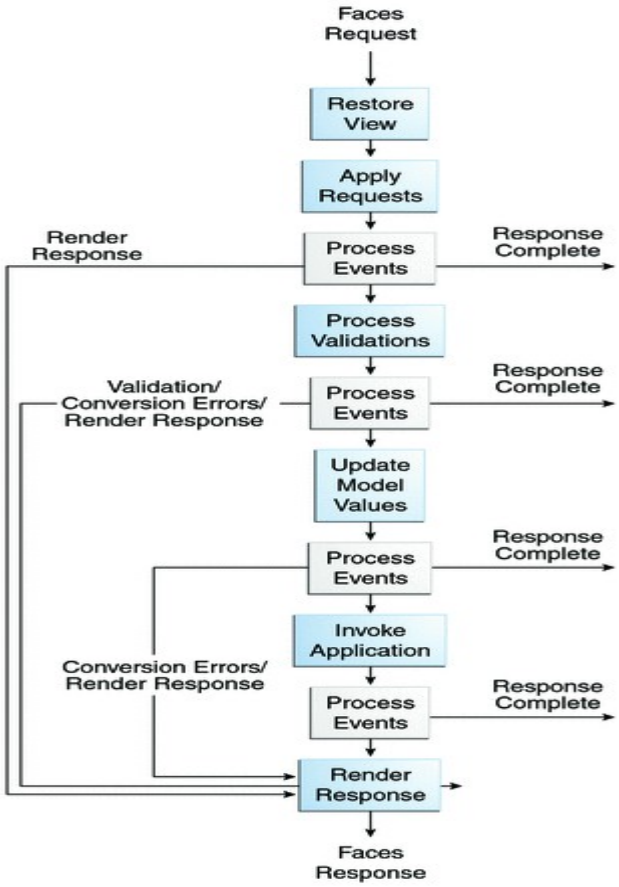


Figura 10: *Ciclo de Vida de un JavaServer Faces*

Fuente:
http://docs.oracle.com/javaee/6/tutorial/doc/figures/jsfintro-lifecycle.gif

1. ***Restore View o Fase de Recuperación:*** Durante esta fase, JavaServer Faces construye la vista de la página, enlaza los controladores de eventos y validadores con los componentes correspondientes, y almacena el resultado en el contexto (FacesContext), donde se contiene toda la información necesaria para procesar la petición.

   Si la petición se realiza por primera vez, se genera una vista vacía y se avanza directamente hasta la fase de renderizado.

   Si no es la primera vez que se solicita, ya existe una vista para la página. Entonces, JSF recupera la vista en función de la información de estado - que puede estar tanto en el cliente o en el servidor.

2. ***Apply Request Values o Fase de aplicación de los valores de la petición:*** Una vez se ha restaurado el árbol de componentes, cada uno de ellos obtiene su nuevo valor a partir de los parámetros pasados en la petición. Si fallara la conversión de este valor, se genera un mensaje de error que se pone en una cola dentro del contexto. Este mensaje se mostrará durante la fase de renderizado, junto a los generados, de haberlos, durante la fase de validación.

   Si durante esta fase, algún método o evento llama a `renderResponse`, se salta a la fase de renderizado.

   Si algún evento ha sido detectado durante esta fase, JavaServer Faces los envía a los manejadores correspondientes.

   Si algún componente en la página lo solicita (a través de un atributo

especial llamado `immediate` - inmediato), entonces se realizan la validación, conversión y eventos asociados, en esta misma fase. En este punto, se realiza una llamada a `FacesContext.responseComplete` si la aplicación necesita redirigir a otro recurso, o enviar una respuesta no-faces.

3. ***Proccess Validations o Fase de Validación:*** Durante esta fase se realiza la validación de los valores de los componentes.
   Si algún valor resulta ser inválido, se añade un mensaje de error al contexto y se avanza hasta la fase de renderizado, de forma que se muestre el mensaje de error.

   De nuevo, puede llamarse a renderResponse para ir directamente a la fase de renderizado.

   Al igual que en la fase anterior, puede llamarse a `FacesContext.responseComplete` si la respuesta es no-faces.
   Si se genera algún evento, se envía a su manejador.

4. ***Update Model Values o Fase de Actualización de Valores:*** Una vez se ha determinado que los valores son válidos, se recorre el árbol de componentes y se sincronizan las propiedades del objeto en el lado del servidor con los valores del componente.
   Si durante esta fase, algún método o evento llama a `renderResponse`, se salta a la fase de renderizado.

   Si algún evento ha sido detectado durante esta fase, JSF los envía a los manejadores correspondientes.
   Si algún componente en la página lo solicita (a través de un atributo

especial llamado `immediate` - inmediato), entonces se realizan la validación, conversión y eventos asociados, en esta misma fase.

5. ***Invoke Application o Fase de Invocación de la Aplicación:*** En esta etapa JSF maneja cualquier evento en el nivel de aplicación, como puede ser el envío de un formulario a otra página.

   Si la aplicación necesita redirigir a otro recurso, o generar una respuesta no-faces, se puede llamar a `FacesContext.responseComplete`.

   Si la vista que está siendo procesada fue reconstruida a partir de la información de estado de una petición previa, y si un componente ha provocado un evento, estos se mandan a sus manejadores.

6. ***Render Response o Fase de renderización de la Respuesta:*** JavaServer Faces delega el renderizado al contenedor JavaServer Pages si la aplicación lo usa. Si es una petición inicial, los componentes presentes en la página se irán añadiendo al árbol de componentes conforme el contenedor JavaServer Pages ejecuta la página. Si no lo es, los componentes ya están presentes en el árbol, por lo que no es necesario volver a añadirlos. En ambos casos, se renderizarán los componentes conforme se procesa la página.

   Si se han encontrado errores, se mostrarán en la página si procede. Una vez se ha renderizado la vista, el estado de la respuesta es almacenado, de forma que pueda recuperarse de nuevo más adelante.

2.4. Patrón de Diseño MVC

Es un patrón de diseño de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de negocio en tres componentes distintos (Modelo, Vista y Controlador). El patrón de llamada y retorno, se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página, el modelo es el Sistema de Gestión de Base de Datos y la lógica de negocio y el controlador es el responsable de recibir los eventos de entrada desde la vista.

2.4.1. Componentes[10]

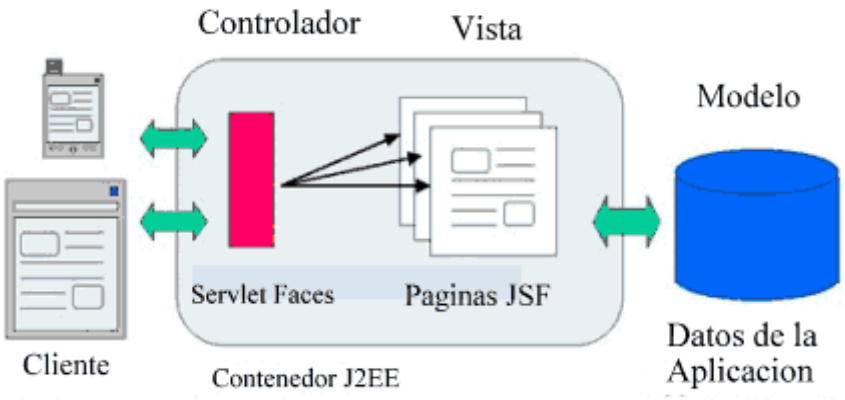


Figura 11: *Patrón de diseño MVC*

Fuente:
http://aragorn.pb.bialystok.pl/~dmalyszko/PSS_Project/JavaServer%20Faces_pliki/image002.jpg

a) Modelo

Esta es la representación especifica de la información con la cual el sistema opera. En resumen, el modelo se limita a lo relativo de la

10 **JAVA SE APPLICATION DESIGN WITH MVC**
http://www.oracle.com/technetwork/articles/javase/index-142890.html
2013-06-09

vista y su controlador facilitando las presentaciones visuales complejas. El sistema también puede operar con más datos no relativos a la presentación, haciendo uso integrado de otras lógicas de negocio y de datos afines con el sistema modelado.

b) Vista

Este presenta el modelo en un formato adecuado para interactuar usualmente la interfaz de usuario.

c) Controlador

Este responde a eventos, usualmente acciones del usuario e invoca peticiones al modelo y probablemente genera una vista.

2.4.2. Beneficios

a) Definir APIs para validación de la entrada, incluyendo soporte cliente-side validation.

b) Especificar un modelo para la internacionalización y localización de la GUI.

c) Generación automática de salida apropiada para el cliente (tomando en cuenta configuración data, browser version, etc).

2.4.3. Funcionamiento

a) El usuario interactua con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc).

b) 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.

c) El controlador accede al modelo, actualizándolo, posiblemente modificándose de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de compras del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comandos que encapsula las acciones y simplifica su extensión.

d) 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 dirección 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.

e) La interfaz de usuario espera nuevas interacciones del usuario,

comenzando el ciclo nuevamente.

2.5. Plataforma Java EE 6 y el Patrón de Diseño MVC

Java EE es una tecnología que abarca dentro de su especificación un gran conjunto de framework, entre el cual se encuentra JSF (JavaServer Faces), este framework está implementado con el patrón de diseño MVC para aprovechar todas las funcionalidades que ofrece el patrón de diseño, así también puede acoplarse a un conjunto de tecnologías.

Dentro de la especificación para JavaServer Faces incluye:

a) Un conjunto de APIs para representar componentes de la interfaz de usuario, maneja su estado, evento, entrada, accesibilidad, internacionalización.

b) Conjunto estándar de Interfaces de Usuario.

2.6. Servidores de Aplicaciones[11]

El concepto de servidor de aplicaciones está relacionado con el concepto de sistemas distribuidos. Un sistema distribuido, en oposición monolítica, permite mejorar tres aspectos fundamentales en una aplicación, *la alta disponibilidad, la escalabilidad y el mantenimiento*. En un sistema monolítico un cambio en las necesidades del sistema provoca un colapso y la adaptación a dicho cambio puede resultar catastrófica. Se va a ver las características de los servidores de aplicaciones:

- La **alta disponibilidad** hace referencia a que un sistema debe estar

11 **INTRODUCCION A LOS SERVIDORES DE APLICACIONES**
http://www.jtech.ua.es/j2ee/2003-2004/abierto-j2ee-2003-2004/sa/sesion1-apuntes.htm
2013-04-19

funcionando las 24 horas del día los 365 días del año. Para poder alcanzar está característica es necesario el uso de técnicas de balanceo de carga y de recuperación ante fallos.

- La **escalabilidad** es la capacidad de hacer crecer un sistema cuando se incrementa la carga de trabajo. Cada máquina tiene una capacidad finita de recursos y por lo tanto solo puede servir un número limitado de peticiones. Si, por ejemplo, se tiene una tienda que incrementa la demanda de servicio, se debe ser capaces de incorporar nuevas máquinas para dar servicio.
- El **mantenimiento** tiene que ver con la versatilidad a la hora de actualizar, depurar de fallos y mantener un sistema. La solución al mantenimiento es la construcción de la lógica de negocio en unidades reusables y modulares.

2.6.1.1. Servidor de Aplicaciones para la plataforma Java EE

El estándar de la plataforma Java EE 6 permite el desarrollo de aplicaciones de empresas de una manera sencilla y eficiencia. Una aplicación desarrollada con la tecnología Java EE, permite ser desplegadas en cualquier servidor de aplicaciones o servidor web que cumpla con el estándar. Un servidor de aplicaciones es una implementación de la especificación de la plataforma Java EE.

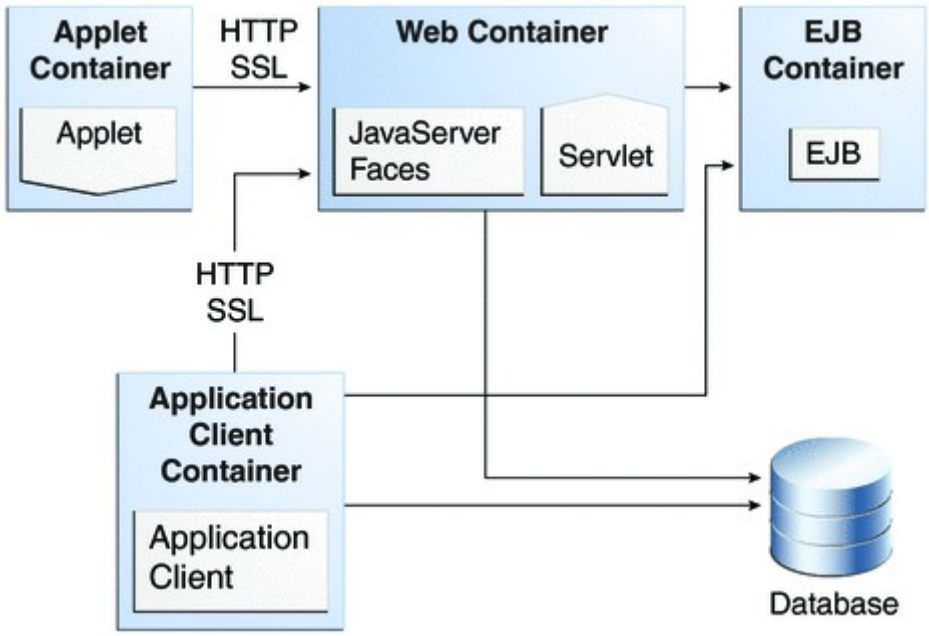


Figura 12: *Arquitectura de la Plataforma Java, Enterprise Edition 6*

Fuente: http://kvgrao.com/jee/figures/overview-architecture-containers.gif

Se define a continuación algunos de los conceptos que se muestra en la figura anterior.

- ***Cliente Web:*** es usualmente un navegador que interactúa con el contenedor web haciendo uso de HTTP. Recibe páginas XHTML y puede ejecutar Applets y código JavaScript.
- ***Aplicaciones Cliente:*** Son clientes que no se ejecutan dentro de un navegador y puede utilizar cualquier tecnología para comunicarse con el contenedor web o directamente con la base de datos.
- ***Contenedor Web:*** Es lo que comúnmente se denomina servidor web. Es la parte visible del servidor de aplicaciones. Utilizan los protocolos HTTP(S) para la comunicación.

- ***Servidor de Aplicaciones:*** Proporciona servicios que soportan la ejecución y disponible de las aplicaciones desplegadas. Es el corazón de un gran sistema distribuido.

Frente a la tradicional estructura de dos capas de un servidor (Cliente – Servidor) un servidor de aplicaciones proporciona una estructura en tres capas que permite estructurar nuestro sistema de forma más eficiente. Una pequeña aplicación que acceda a una base de datos no muy compleja y que no sea distribuida probablemente no necesitará un servidor de aplicaciones, tan solo con un servidor de contenedor web sea suficiente.

Java EE Platform Technology	Java Enterprise Edition 6	
	Full Profile	Web Profile
Web Application Technologies		
JSR 315: Java Servlet 3.0	X	X
JSR 314: JavaServer Faces(JSF) 2.0	X	X
JSR 245: JavaServer Pages 2.2 and Expression Language (EL (1.2)	X	X
JSR-52: A Standard Tag Library for JavaServer Pages 1.2	X	X
JSR-45: Debugging Support for Other Languages 1.0	X	X
Enterprise Application Technologies		
JSR 299: Contexts and Dependency Injection for the Java EE Platform 1.0	X	X
JSR 330: Dependency Injection for Java	X	X
JSR 318: Enterprise JavaBeans 3.1	X	X (Lite)
JSR 317: Java Persistence API 2.0	X	X
JSR 250: Common Annotations for the Java Platform 1.1	X	X
JSR 907: Java Transaction API (JTA) 1.1	X	X
JSR 303: Bean Validation 1.0	X	X
JSR 322: Java EE Connector Architecture 1.6	X	--
JSR 914: Java Message Service (JMS) API 1.1	X	--
JSR 919: JavaMail 1.4	X	--
Web Services Technologies		
JSR 311: JAX-RS: The Java API for RESTful Web Services 1.1	X	--
JSR 109: Implementing Enterprise Web Services 1.3	X	--
JSR 224: Java API for XML-Based Web Services (JAX-WS) 2.2	X	--
JSR 181: Web Services Metadata for the Java Platform	X	--
JSR 101: Java APIs for XML based RPC 1.1	X	--
JSR 93: Java API for XML Registries 1.0 (JAXR) 1.0*	X	--
Management and Security Technologies		
JSR 196: Java Authentication Service Provider Interface for Containers 1.0	X	--
JSR 115: Java Authorization Contract for Containers 1.4	X	--
JSR 77: J2EE Management 1.1*	X	--
JSR 88: Java EE Application Deployment 1.2*	X	--

Figura 13: *Especificación de la Plataforma Java, Enterprise Edition*

Fuente:
https://docs.jboss.org/author/download/attachments/66322959/JBossAS7-JavaEE.png

Como se ha comentado, un servidor de aplicaciones es una implementación de la especificación de la plataforma Java EE 6, como se muestra en la figura 13. Existen diversas implementaciones de software libre, cada una con sus propias características que la pueden hacer más atractiva en el desarrollo de un determinado sistema. Algunas de las implementaciones más utilizadas son las siguientes:

- GlassFish 3.1.x
- JBoss AS 7.1.x

2.7. Servidores de Aplicación GlassFish y JBoss AS

Tanto GlassFish en su versión 3,0 o posterior así como JBoss AS en su versión 7,1 o posterior cumplen con la certificación de la Plataforma Java, Enterprise Edition 6 Full Profile como se muestra en la Figura 13: Especificación de la Plataforma Java, Enterprise Edition a los cuales se les considera Servidores de Aplicación Certificados.

Tanto GlassFish como JBoss AS son servidores de aplicaciones que implementan cada una de las especificaciones de la Plataforma Java Enterprise Edition 6, estos servidores son distribuidos bajo licencias de software libre.

2.7.1. Características de los servidores de aplicación

1. Los servidores de aplicación permite la comunicación entre diversos servicios.
2. Los servidores de aplicación brindan a los desarrolladores una API para disminuir el desarrollo y dedicarse a la parte de la lógica del

negocio.

3. Los servidores de aplicación también brindan soporte a una gran variedad de estándares, tales como HTML, XML, JDBC, SSL, Web Services, etc.

4. Facilidad de instalación y administración

2.8. Descripción de la Metodología SCRUM

Scrum es una metodología de desarrollo muy simple, que requiere trabajo duro, porque no se basa en el seguimiento de un plan, sino en la adaptación continua a las circunstancias de la evolución del proyecto.

Como método ágil:

- Es un modo de desarrollo adaptable, antes que predictivo.
- Orientado a las personas, más que a los procesos.
- Emplea el modelo de construcción incremental basado en iteraciones y revisiones.

Comparte los principios estructurales del desarrollo ágil: a partir del concepto o visión de la necesidad del cliente, construye el producto de forma incremental a través de iteraciones breves que comprenden fases de especulación, exploración y revisión. Estas iteraciones se repiten de forma continua hasta que el cliente da por cerrado el producto.

Se comienza con la visión general del producto, especificando y dando detalle a las funcionalidades o partes que tienen mayor prioridad de negocio y que pueden llevarse a cabo en un período de tiempo breve.

Estas iteraciones son la base del desarrollo ágil y Scrum gestiona su evolución en reuniones breves diarias donde todo el equipo revisa el trabajo realizado el día anterior y el previsto para el siguiente.

2.8.1. Ciclo de desarrollo ágil[12]

El desarrollo ágil parte de la visión, del concepto general del producto, y sobre ella el equipo produce de forma continua incrementos en la dirección apuntada por la visión; y en el orden de prioridad que necesita el negocio del cliente. Los ciclos breves de desarrollo, se denominan iteraciones y se realizan hasta que se decide no evolucionar más el producto.

Este esquema está formado por cinco fases:

1. Concepto
2. Especulación
3. Exploración
4. Revisión
5. Cierre

12 **PALACIOS J.,** Scrum Manager: Gestión de Proyectos., 2ª, ed., Safe Creative., 2010., Pg 44

Concepto

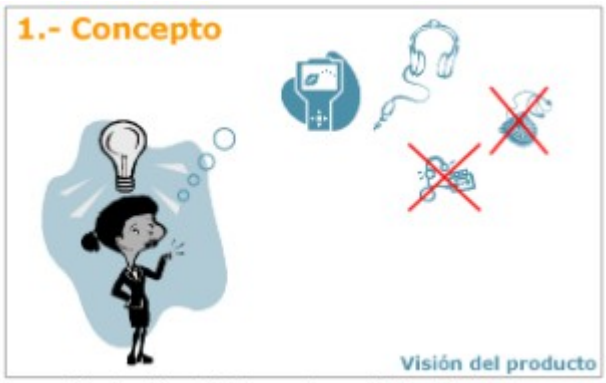


Figura 14: *Concepto o visión del cliente*

Fuente: Imagen tomado del libro Scrum Manager Revisión1.4

En esta fase se crea la visión del producto y se determina el equipo que lo llevará a cabo.

Partir sin una visión genera esfuerzo baldío. La visión es un factor crítico para el éxito del proyecto. Se necesita tener el concepto de lo que se quiere, y conocer el alcance del proyecto.

Es además una información que se debe compartir todos los miembros del equipo

Especulación

Figura 15: *Especulación o aportación de todo el equipo*

Fuente: Imagen tomado del libro Scrum Manager Revisión1.4

Una vez que se sabe qué hay que construir, el equipo especula y formula hipótesis basadas en la información de la visión, muy general e insuficiente para determinar las implicaciones de un desarrollo (requisitos, diseño, costes...).

En esta fase se determinan las limitaciones impuestas por el entorno de negocio: costes y agendas principalmente, y se cierra la primera aproximación de lo que se puede producir.

La gestión ágil investiga y construye a partir de la visión del producto.

La fase de especulación se repite en cada iteración y teniendo como referencia la visión y el alcance del proyecto consiste en:

- Desarrollo y revisión de los requisitos generales.
- Mantenimiento de una lista con las funcionalidades esperadas.
- Mantenimiento de un plan de entrega: fechas en las que se

necesitan las versiones, hitos e iteraciones del desarrollo. Este plan refleja ya el esfuerzo que consumirá el proyecto durante el tiempo.

- En función de las características del modelo de gestión y del proyecto puede incluir también una estrategia o planes para la gestión de riesgos.

Si las exigencias formales de la organización lo requieren, también se produce información administrativa y financiera.

Exploración

Se desarrolla un incremento del producto, que incluye las funcionalidades determinadas en la fase anterior.

Revisión

Figura 16: *Revisión*

Fuente: Imagen tomado del libro Scrum Manager Revisión1.4

Equipo y usuarios revisan lo construido hasta ese momento. Trabajan y operan con el producto real contrastando su alineación con el objetivo.

Cierre

Al llegar a la fecha de entrega de una versión de producto (fijada en la fase

de concepto y revisada en las diferentes fases de especulación), se obtiene el producto esperado.

Posiblemente éste seguirá en el mercado y por emplear la gestión ágil, es presumible que se trata de un producto que necesita versiones y mejoras frecuentes para no quedar obsoleto. El cierre no implica el fin del proyecto.

Lo que se denomina “mantenimiento” supondrá la continuidad del proyecto en ciclos incrementales hacia la siguiente versión para ir acercándose a la visión del producto.

2.8.2. Control de la evolución del proyecto

Scrum controla de forma empírica y adaptable la evolución del proyecto, a través de las siguientes prácticas de la gestión ágil.

2.8.2.1. Revisión de las Iteraciones

Al finalizar cada iteración (sprint) se lleva a cabo una revisión con todas las personas implicadas en el proyecto. Es por tanto la duración del sprint, el período máximo que se tarda en reconducir una desviación en el proyecto o en las circunstancias.

2.8.2.2. Desarrollo incremental

Las personas implicadas no trabajan con diseños o abstracciones.

El desarrollo incremental implica que al final de cada iteración se dispone de una parte de productos operativa, que se puede inspeccionar y evaluar.

2.8.2.3. Desarrollo evolutivo

Los modelos de gestión ágil se emplean para trabajar en entornos de incertidumbre e inestabilidad de requisitos.

Intentar predecir en las fases iniciales cómo será el resultado final y sobre dicha predicción desarrollar el diseño y arquitectura del producto no es realista, porque las circunstancias obligarán a remodelar muchas veces.

¿Para qué predecir los estados finales de la arquitectura o del diseño si van a estar cambiando?

Scrum considera a la inestabilidad como una premisa y se adoptan técnica de trabajo para permitir la evolución sin degradar la calidad de la arquitectura que también evoluciona durante el desarrollo.

Durante el desarrollo se genera el diseño y la arquitectura final de forma evolutiva.

Scrum no lo considera como producto que deban realizarse en la primera base del proyecto.

2.8.2.4. Auto-organización

En la ejecución de un proyecto son muchos los factores impredecibles en todas las áreas y niveles.

La gestión predictiva confía la responsabilidad de su resolución al gestor de proyectos. En Scrum los equipos son auto-organizados (no auto-dirigidos), con margen de decisión suficiente para tomar las decisiones que consideren oportunas.

2.8.2.5. Colaboración

Las prácticas y el entorno de trabajo ágiles facilitan la colaboración del equipo. Ésta es necesaria, porque para que funcione la auto-organización como un control eficaz cada miembro del equipo debe colaborar de forma abierta con los demás, según sus capacidades y no según su rol o su puesto.

2.8.2.6. Visión general del proceso

Scrum denomina "sprint" a cada iteración de desarrollo y según las características del proyecto y las circunstancias del sprint puede determinarse una duración desde una hasta dos meses, aunque no suele ser recomendable hacerlos de más de un mes. El sprint es el núcleo central que proporciona la base de desarrollo iterativo e incremental.

2.8.3. Los elementos

Los elementos centrales del ciclo ágil Scrum son:

- Pilas del producto (Product Backlog): Lista de funcionalidades que necesita el cliente.
- Pilas del sprint (Sprint Backlog): Lista de tareas que se realizan en un sprint.
- Incremento: Parte del sistema desarrollando en un sprint.

Figura 17: *Elementos centrales de Scrum*

Fuente: Imagen tomado del libro Scrum Manager Revisión1.4

A continuación se describe estos tres elementos. Los dos primeros forman los requisitos del sistema y el tercero es valor que se le entrega al cliente final de cada sprint.

Cada incremento es una parte del producto completo terminado y operativo.

No se deben considerar como incrementos: prototipos o módulos pendientes de pruebas o de integración.

2.8.4. Pilas del producto: los requisitos del cliente

La pila del producto es el inventario de funcionalidad, mejoras, tecnologías y corrección de errores que deben incorporarse al producto a través de las sucesivas iteraciones de desarrollo.

Representa todo aquello que esperan los clientes, usuarios y en general los interesados. Todo lo que suponga un trabajo que debe realizar el equipo tiene que estar reflejando en esta pila.

Estos son algunos ejemplos de posibles entradas de la pila del producto:

- Permitir a los usuarios la consulta de las obras públicas por un determinado autor.
- Reducir el tiempo de instalación del programa.
- Mejorar la escalabilidad del sistema.
- Permite la consulta de una obra a través de un API web.

A diferencia de un documento de requisitos del sistema, la pila del producto nunca se da por completada; está en continuo crecimiento y evolución.

Habitualmente se comienza a elaborar con el resultado de una reunión de “fertilización cruzada” o brainstorming; o un proceso de “Exportación” donde colabora todo el equipo a partir de la visión del propietario del producto.

El formato de la visión no es relevante. Según los casos, puede ser una presentación informal del responsable del producto, un informe de requisitos del departamento de marketing, etc.

Sí que es importante sin embargo disponer de una visión real, comprendida y compartida por todo el equipo.

La pila evolucionará de forma continua mientras el producto esté en el mercado, para darle valor de forma continua y mantenerlo útil y competitivos.

Para dar comienzo al desarrollo se necesita una visión de los objetivos de negocio que se quieren conseguir con el proyecto, comprendida y conocida por todo el equipo y elementos suficientes en la pila para llevar a cabo el primer sprint.

2.8.4.1. Formato de la pila del producto

Id	Prioridad	Descripción	Est.	Por
1	Muy alta	Plataforma tecnológica	30	AR
2	Muy alta	Interfaz usuario	40	LR
3	Muy alta	Un usuario se registra en el sistema	40	LR
4	Alta	El operador define el flujo y textos de un expediente	60	AR
5	Alta	Etc...	999	XX

Figura 18: *Ejemplo de pila de producto*

Fuente: Imagen tomado del libro Scrum Manager Revisión1.4

El desarrollo ágil prefiere la comunicación directa, a la comunicación con documentos.

La pila del producto no es un documento de requisitos, sino una herramienta de referencia para el equipo.

Si se emplea formato de lista, es recomendable que al menos incluya la siguiente información en cada línea:

- Identificador único de la funcionalidad o trabajo
- Descripción de la funcionalidad
- Campo o sistema de priorización
- Estimación

Dependiendo del tipo de proyecto, funcionamiento del equipo y la organización, pueden resultar aconsejables otros campos:

- Observaciones
- Criterio de validación
- Persona asignada
- N° de Sprint en el que se realiza

- Módulos del sistema al que pertenece.

2.8.5. Pilas del sprint

La pila del sprint, es la lista que descompone las funcionalidades de la pilas del producto en las tareas necesarias para construir un incremento: una parte completa y operativa del producto.

La realiza el equipo durante la reunión de planificación del sprint, asignado cada tarea a una persona e indicando en la misma lista cuánto tiempo falta aún para que la termine.

Es útil porque descompone el proyecto en unidades de tamaño adecuado para determinar el avance diario e identificar riesgos y problemas sin necesidades de procesos complejos de gestión. Es también una herramienta de soporte para la comunicación directa del equipo.

2.8.5.1. Condiciones

- Realizada de forma conjunta por todos los miembros del equipo.
- Cubre todas las tareas identificadas por el equipo para conseguir el objetivo del sprint.
- Sólo el equipo lo puede modificar durante el sprint.
- El tamaño de cada tarea está en un rango de 2 a 16 horas de trabajo.
- Es visible para todo el equipo. Idealmente en una pizarra o pared en el mismo espacio físico donde trabaja el equipo.

2.8.5.2. Formato y soporte

SPRINT	INICIO	DURACIÓN
1	1-mar-07	12

			J
			1-mar
			23
			276
SPRINT BACKLOG			
Tarea	Estad	Responsal	
Descripción de la tarea 1	Terminada	Luis	16
Descripción de la tarea 2	Terminada	Luis	12
Descripción de la tarea 3	Terminada	Luis	4
Descripción de la tarea 4	Terminada	Elena	8
Descripción de la tarea 5	Terminada	Elena	16
Descripción de la tarea 6	Terminada	Elena	6
Descripción de la tarea 7	Terminada	Antonio	16
Descripción de la tarea 8	Terminada	Antonio	16
Descripción de la tarea 9	Terminada	Antonio	12
Descripción de la tarea 10	En curso	Luis	12
Descripción de la tarea 11	Pendiente	Luis	8

Figura 19: *Ejemplo de pila de sprint en una hoja de cálculo*

Fuente: Imagen tomado del libro Scrum Manager Revisión1.4

Tres son las opciones:

- Hoja de cálculo.
- Pizarra física o pared.
- Herramienta colaborativa o de gestión de proyectos.

Y sobre la que mejor se adecua a las características del proyecto, oficina y equipo, lo apropiado es diseñar el formato más cómodo para todos, teniendo en cuenta los siguientes criterios:

- Incluye la información: lista de tareas, persona responsable de cada una, estado en el que se encuentra y tiempo de trabajo que queda para completarla.
- Sólo incluye la información estrictamente necesaria.
- El medio y modelo elegido es la opción posible que más facilita la consulta y comunicación diaria y directa del equipo.
- Sirve de soporte para registrar en cada reunión diaria del sprint, el

tiempo que le queda a cada tarea.

2.8.6. El Incremento

El incremento es la parte de producto producida en un sprint y tiene como característica:

- Está completamente terminada y operativa, en condiciones de ser entregada al cliente final.

No se trata por tanto de módulo o parte a falta de pruebas o documentación.

Idealmente en el desarrollo ágil:

- Cada funcionalidad de la pila del producto se refiere a funcionalidades entregables, no a trabajos internos del tipo “diseño de base de datos”
- Se produce un “incremento” en cada iteración.

Sin embargo suele ser una excepción habitual el primer sprint. En el que objetivos del tipo “contrastar la plataforma y el diseño” pueden ser normales e implican trabajos de diseño o desarrollo de prototipos para probar la solvencia de la plataforma que se va a emplear, etc.

Teniendo en cuenta está excepción habitual:

Incremento es:

Parte de producto realizada en un sprint y potencialmente entregable: TERMINADA Y PROBADA.

Si el proyecto o el sistema requiere documentación o procesos de

validación y verificación documentados o con niveles de independencia que implican procesos con terceros, éstos también tienen que estar realizados para considerar que el producto está “terminado”.

2.8.7. Las reuniones

Scrum realiza el seguimiento y la gestión del proyecto a través de las tres reuniones que forman parte del modelo:

- Planificación del sprint
- Seguimiento del sprint
- Revisión del sprint

Figura 20: *Reuniones en Scrum*

Fuente: Imagen tomado del libro Scrum Manager Revisión1.4

2.8.7.1. Planificación del sprint

En esta reunión se toman como base las prioridades y necesidades de negocio del cliente y se determina cuáles y cómo van a ser las

funcionalidades que incorporará el producto tras el siguiente sprint.

En realidad es una reunión que consta de dos partes:

- En la primera: que puede tener una duración de una a cuatro horas, se decide qué elementos de la pila del producto se van a desarrollar.
- En la segunda: se desglosan estos para determinar las tareas necesarias, estimar el esfuerzo para cada una y asignarlas a las personas del equipo.

2.8.7.2. Seguimiento del sprint

Reunión diaria breve, de no más de 15 minutos, en la que cada miembro del equipo dice las tareas en las que está trabajando, si se ha encontrado o prevé encontrarse con algún impedimento y actualización sobre la pila del sprint las ya terminadas o los tiempos de trabajo que les quedan.

2.8.7.3. Revisión del sprint

Reunión realizada al final del sprint en la que, con una duración máxima de 4 horas, el equipo presenta al propietario del producto, clientes, usuario, gestores, etc. El incremento concluido en el sprint.

2.9. Benchmarking

2.9.1. Etimología de benchmarking[13]

Originalmente la expresión *"Benchmark"* proviene de la topografía. Es una marca que hace topógrafos en una roca o un poste de concreto, para comparar niveles.

13 **SOLFA F.,** Benchmarking en el sector público., La Plata-Argentina, 2012., Pg. 9

El benchmarking es un término que fue utilizado originalmente por los agrimensores para comparar alturas. Hoy, sin embargo, el benchmarking tiene un significado más restringido en el léxico de gestión, siendo el punto de comparación de la mejor práctica del sector.

2.9.2. La técnica de benchmarking orígenes y definiciones

El benchmarking aparece en Estados Unidos a finales de la década del setenta, a partir de la necesidad de la Compañía Xerox de poder entender y superar sus desventajas competitivas[14].

Posteriormente, otras organizaciones empresariales se destacaron al implementar con éxito benchmarking, incluyendo a Ford Motor Company, Alcoa, Millken, AT&T, IBM, Johnson & Johnson, Kodak, Motorola y Texas Instruments; tornándose casi obligatorio para cualquier organización que desee mejorar sus productos, servicios, procesos y resultados.

La denominación del benchmarking, se atribuye a la publicidad de Camp 1991[15] donde trata la aplicación de Xerox, como una técnica de autoevaluación y búsqueda de las mejores prácticas con el objetivo de mejorar la calidad de sus procesos.

Esta publicación, coincidió con la distinción del Premio Nacional de Calidad Malcolm Baldrige a la compañía Xerox, que consiguió su liderazgo en la calidad a partir de las técnicas de benchmarking. Este premio, incluía entre los criterios de evaluación, la implementación de información actualizada y el desarrollo de evaluaciones comparativas, una de las primeras fases de

14 Quizás el primer antecendente occidental del método, se romenta a la Segunda Guerra Mundial, cuando entre empresas estadounidenses, se convirtió en una práctica común compararse entre sí a fin de determinar patrones para pagos, carga de trabajo, seguridad, higiene y otros factores conexos.

15 Originalmente, la obra fue publicada en ingles en 1989, como: “Benchmarking: the Search for Industry Best Practice that Lead to Superior Perfomance”, Quality Press, American Society for Quality Control, Milwaukee.

lo que considera hoy benchmarking.

Distintos autores definen al benchmarking como un proceso de evaluación comparada, continua y sistemática entre organizaciones, de procesos, productos y servicios; con el fin de implementar mejoras en una organización. (Spendolini, 1994)

La comisión Directiva del Internacional Benchmarking Clearinghouse del American Productivity & Quality Center (APQC), define al benchmarking como "un proceso de evaluación continuo y sistemático; un proceso mediante el cual se analizan y comparan permanentemente los procesos empresariales de una organización frente a los procesos de las compañías líderes en cualquier parte del mundo, a fin de obtener información que pueda ayudar a la organización en su rendimiento. (Citado por Montero y Oreja, 2010: p. 182)

Benchmarking es una estrategia independiente de gestión e integra evolucionadamente en un conjunto de técnicas de calidad. Es también una técnica de innovación de la gestión. (Cohen y Eimicke, 1996; Clemente y Balmaseda, 2010)

2.9.3. Los beneficios de su utilización[16]

Las organizaciones vienen utilizando el benchmarking con diferentes fines. Algunas ubican al benchmarking como parte de un proceso general que busca mejorar a la organización. Otras lo conciben como un mecanismo continuo para mantenerse actualizadas (Spendolini, 1997).

Es una técnica muy eficiente para introducir mejoras en las organizaciones,

16 **SOLFA Federico del Giorgio,** Benchmarking en el sector público. Mayo 2012. Pg. 11

ya que pueden incorporarse y adaptarse a procesos cuya efectividad ya ha sido probada por otras organizaciones. Por esta razón, ayuda a las organizaciones a introducir mejoras rápidamente.

El benchmarking es una técnica relativamente baja en tecnología, de bajo costo y rápida respuesta, que cualquier organización puede adoptar. También pareciera tener el suficiente sentido común, como para que sea fácil de entender tanto para directivos, gerentes, trabajadores, proveedores, clientes, como para los medios de comunicación y público en general. (Cohen y Eimicke, 1995 y 1996; Cohen et al., 2008).

La innovación es uno de los beneficios directos que se obtienen a partir de las prácticas de benchmarking y tiene incidencia directa en las formas del hacer, a partir de la incorporación de nuevas concepciones de un tema, idea o aplicación concreta. (Clemente y Balmaseda, 2010).

2.9.4. Principales características

El benchmarking puede entenderse como un mecanismo interno clave, para el desarrollo de la cultura de la mejora continua en las organizaciones. El potencial de esta técnica, depende principalmente de su utilización continua; el benchmarking no es sólo un proceso que se lleva a cabo solo una vez, sino que es un proceso continuo y constante.

Para llevar a cabo procesos de benchmarking, deben medirse los procesos propios y el de otras organizaciones para poder compararlos. Las comparaciones deben realizarse con organizaciones líderes, lo que cambia la práctica de la comparación interna por una comparación en base a estándares externos, derivados de organizaciones reconocidas como

líderes en el sector o en el proceso.

2.9.5. La importancia de la aplicación del benchmarking en el sector público.[17]

Benchmarking, es una técnica de gestión, que básicamente comprende un proceso continuo de medición de productos, servicios y tecnologías de producción de una determinada organización, para comparar con los de una organización modelo.

El benchmarking, es una de las técnicas de gestión de relativo éxito, que ha sido muy difundida y utilizada en el sector privado. Desde hace algunos años, se vienen realizando aplicaciones en el sector público de manera sectorizada.

En la última década, diferentes gobiernos de Europa y América, vienen desarrollando exitosamente aplicaciones más integrales de metodologías de benchmarking, en diferentes áreas temáticas del sector público: territorios, empresas, servicios públicos, universidades, parques científicos, etc.

A partir de su utilización en los países más desarrollados, se ha convertido en un componente elemental de los procesos de regulación y concesión de los servicios públicos. (Chillo, 2010).

Los resultados obtenidos a partir de las aplicaciones de benchmarking en el sector público, han evidenciado un desarrollo de mejores servicios y organizaciones con entornos más eficientes.

Según Fernando Marchitto (2001, 2002 y 2009), quién ha investigado,

17 **SOLFA F.,** Benchmarking en el sector público., La Plata-Argentina, 2012., Pg. 7

desarrollado y aplicado el benchmarking en el sector público, sostiene que para la administración pública, esta técnica podría constituirse el medio adecuado para apropiarse del rol de productores de bienestar para la comunidad, recuperando la eficiencia y la eficacia.

2.9.6. Tipos de benchmarking[18]

Para Camp (1991) existen cuatro tipos de benchmarking: interno, competitivo, funcional y genérico.

En cambio, Spendolini (1994) categoriza tres tipos de benchmarking: interno, competitivo y genérico (funcional), agrupado en una misma categoría al bechmarking genérico y funcional.

El benchmarking interno se centra en la comparación de acciones internas para la identificación de los mejores procesos de la organización.

El competitivo identifica y recaba información sobre procesos, productos y servicios en la competencia directa, para comparar con la propia.

El genérico, identifica y recaba información de igual manera que el competitivo, pero de otras organizaciones que pueden ser o no competidoras.

18 **SOLFA F.,** Benchmarking en el sector público., La Plata-Argentina, 2012., Pg. 12

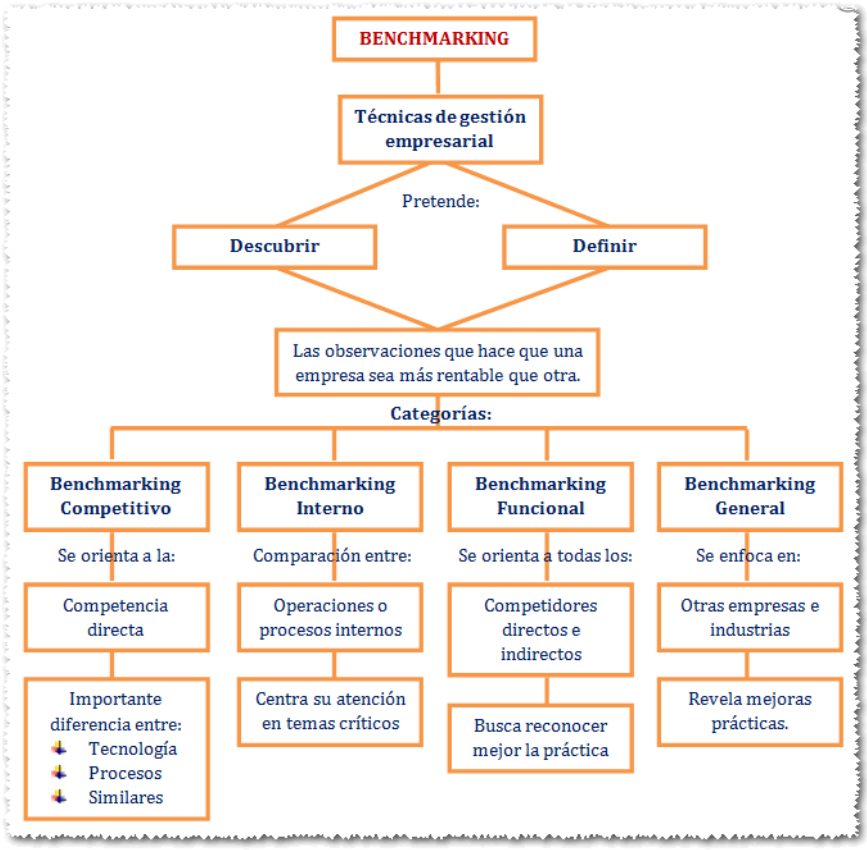


Figura 21: *Mapa de Tipos de Benchmarking*

Fuente: Imagen tomado del libro Benchmarking en el sector público

2.9.6.1. Benchmarking interno

Este tipo de técnica toma como marco de acción a la organización en su conjunto. Quizás sea el más empleado en acciones de calidad institucional, ya que su principal objetivo es identificar los estándares de desarrollo de la organización y de las actividades análogas que pueden existir en diferentes áreas, departamentos, regiones, etc.

Este tipo de técnica es aplicable en grandes organizaciones, donde busca identificar dentro de la misma organización los procesos más eficientes y

eficaces. Pudiendo así elaborar patrones de comparación (benchmark) y tomarse como estándares para iniciar procesos de mejora continua.

El benchmarking interno ayuda a las organizaciones a generar su propio conocimiento y capitalizar en futuras aplicaciones a nivel interno, competitivo o funcional. Además entrena al personal implicado, otorgando motivación por la mejora continua y la excelencia.

Uno de los riesgos del enfoque interno, radica en la posibilidad de que en el proceso de comparación de los métodos internos, no se percibe que éstos sean significativamente menos eficientes que los de otras organizaciones. Es así que el enfoque del benchmarking interno, pueda impedir contar con la visión global que se requiere para comprender los logros de eficiencia alcanzando al externo de la propia organización.

2.9.6.2. Benchmarking competitivo

Este tipo de benchmarking es el más conocido y consiste en identificar, recabar información y analizar procesos, productos y servicios de la competencia, para comparar con los de la organización investigadora.

El benchmarking competitivo, sirve a las organizaciones que persiguen la mejora de sus productos o servicios, dentro del entorno (mercado) en el que participan.

Normalmente, el resto de las organizaciones competidoras, emplean tecnologías u operaciones análogas a las de la propia organización. Identificar estas similitudes, permite comprender cuáles son las ventajas competitivas de las principales organizaciones y aplicarlas a la propia organización como innovaciones.

En este tipo de evaluación, pueden encontrarse algunas limitaciones derivadas de la imposibilidad de acceder a información clave de las operaciones de los competidores, o de las aplicaciones de métodos o diseños del competidor que pueden estar protegidos por registros o patentes.

2.9.6.3. Benchmarking funcional

El benchmarking funcional se enfoca en analizar especialmente funciones y procesos que pertenecen a un mismo sector, pero no están sometidos a competencia.

Se determina funcional, porque refiere a la evaluación comparativa de funciones específicas con otra organización, que posea estándares de excelencia en el área específica donde se realiza el benchmarking.

Es el tipo de benchmarking más aplicado entre los organismos públicos y las grandes empresas de servicios.

2.9.6.4. Benchmarking genérico

Existen acciones y procesos que pueden ser idénticos en organizaciones que pertenecen a sectores y rubros diferentes. Es así, que las áreas o departamentos de contabilidad, facturación, compras, recursos humanos, etc, de organizaciones de diversos sectores, pueden tener similitudes y permitir lógicamente, comparar sus mejores prácticas y adoptar nuevos sistemas o procesos de mejora.

Es por ello, que este tipo de benchmarking, identifica en cualquier tipo de organización (competidores o no), procesos, productos y servicios, con la

finalidad de identificar las mejores prácticas y resultados en un determinado campo específico.

El benchmarking genérico requiere una aptitud de conocimiento en distintos campos de la administración, que permita explorar en distintos sectores otras formas de producir el bien o el servicio, pero con una comprensión acabada del proceso genérico en el que se actúe.

Es el tipo de benchmarking que resulta más dificultoso de incorporar y utilizar en las organizaciones, pero es probablemente el que produzca mayores ventajas competitivas y rendimiento en el largo plazo.

2.9.7. Metodología del benchmarking

Robert Camp (1991 y 1996) ha propuesto la siguiente metodología para la aplicación del benchmarking.

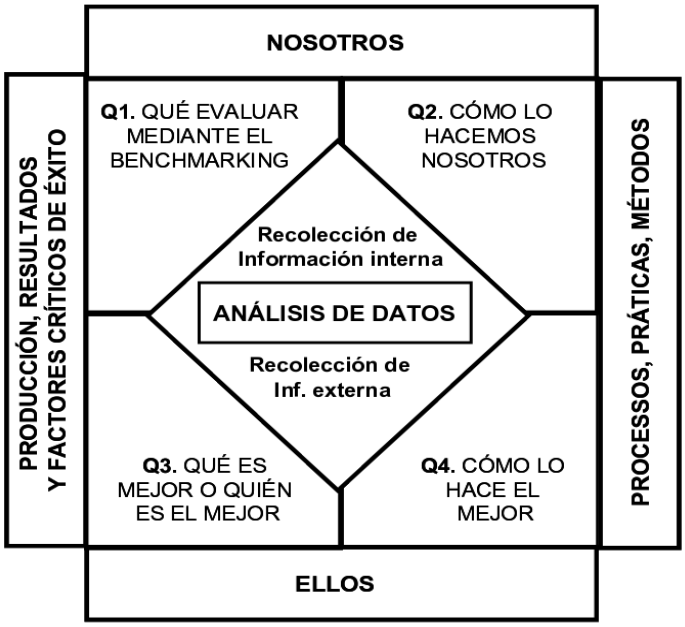


Figura 22: *Cuadrante multidimensional desarrollado por Xerox Corporation*

Fuente: Imagen tomado del libro Benchmarking en el sector público

2.9.7.1. El modelo de Camp

En el modelo adoptado por Robert Camp (1996), se establecen cinco fases con diez pasos:

a) **Fase de Planificación:** El objetivo de esta fase es planificar las investigaciones de benchmarking. Los pasos principales se componen con las acciones tradicionales relacionadas con la planificación (definición de quién, qué y cómo).

   1. Identificar a qué se le realizará el benchmarking (proceso, producto o servicio)

   2. Identificar las organizaciones que pueden ser comparables.

   3. Determinar el método para relevar los datos.

b) **Fase de Análisis:** Una vez determinado el quién, el qué y el cómo, se debe llevar a cabo la recopilación y el análisis de datos. Esta fase debe incluir una comprensión exhaustiva de las prácticas actuales del proceso, así como también de los socios del benchmarking.

   1. Determinar la brecha existente entre el desempeño propio de cada componente.

   2. Planificar los niveles de desempeño futuros.

c) **Fase de Integración:** La integración es la acción que utiliza los resultados del benchmarking para fijar los objetivos y metas operacionales para el cambio.

   1. Comunicar los resultados de benchmarking y obtener la

aceptación.

2. Establecer las metas funcionales.

d) **Fase de Acción:** En esta instancia, se deben convertir en acción los resultados del benchmarking y los principios operacionales basados en estos resultados. Así mismo, es necesario incorporar procesos de evaluación de resultados y evaluar las metas de manera periódica.

1. Desarrollar planes de acción.

2. Implementación de acciones específicas y supervisar el progreso.

3. Recalibrar los patrones de referencia (estándares o benchmarking).

e) **Fase de Madurez:** La madurez es alcanzada cuando se incorporen las mejores prácticas del sector a todos los procesos, asegurando así la superioridad. También se alcanza la madurez, cuando se convierte en una práctica continua, esencial y sistemática del proceso gestión; en otras palabras, cuando se institucionaliza el benchmarking

2.9.8. Sintéticos vs Aplicaciones

Tanto este premio estadounidense como los reconocimientos que otorga la European Foundation for Quality Management (EFQM), contiene cláusulas que le exigen a las organizaciones participantes, compartir información sobre las mejoras de proceso y las estrategias de calidad, lo que posibilita

el acceso a organizaciones menores, de una buena fuente de información sobre la práctica del benchmarking.

- ***Sintéticos:*** están especialmente diseñados para medir el rendimiento de un componente individual de un ordenador. Normalmente llevando el componente escogido a su máxima capacidad.
- ***Aplicaciones:*** herramientas basadas en aplicaciones reales, simulan una carga de trabajo para medir el comportamiento global del equipo.

2.9.9. Bajo nivel vs Alto nivel

- ***Bajo nivel:*** Miden directamente el rendimiento de los componentes, por ejemplo: el reloj de la CPU, los tiempos de la RAM, tiempo de acceso medio al disco duro, latencia, tiempo de cambio de pista etc.
- ***Alto nivel:*** Está enfocado a medir el rendimiento de la combinaciones de componentes/controladores/SO de un aspecto específico del sistema, como por ejemplo: el rendimiento de E/S con ficheros o el rendimiento de una determinada combinación de componentes/controladores/SO/aplicación, por ejemplo: velocidad de compresión de un zip.

2.9.10. Herramientas para realizar el benchmarking

Las herramientas benchmark es también un proceso continuo de medir productos, servicios y prácticas contra competidores más duros o aquellas compañías reconocidas como líderes en la industria.

Para el estudio comparativo se utilizará el paquete sysstat, el cuál es una herramienta benchmark la cual brinda una colección de herramientas para monitorizar el rendimiento de nuestro sistema o parte de nuestros componentes hardware.

Esta herramienta puede ofrecer datos instantáneos de rendimiento, así como almacenados históricamente para futuras referencias.

Especialmente en entornos de servidores, sus datos proporcionan información muy valiosa sobre las posibles carencias y cuellos de botella que puede experimentar el servidor.

2.10. Herramienta ApacheBench[19]

Es una herramienta sencilla que permite testear o estresar a un servidor web desde un cliente, puede medir el rendimiento de su servidor o de cualquier otro. Y una de sus principales ventajas es su extrema sencillez, ya que no hay más que ejecutar el comando `ab` con un par de parámetros para obtener información muy útil como se muestra en la siguiente figura.

```
ab -n 100 -c 10 http://www.genbetadev.com/
```

Figura 23: *Comando para estresar un servidor web*

Fuente:
http://www.genbetadev.com/herramientas/apachebench-una-sencilla-herramienta-para-testear-servidores-web

19 **ab – Apache HTTP server benchmarking tool** http://httpd.apache.org/docs/2.2/programs/ab.html 2013-06-09

Con el comando anterior se realiza 100 llamadas a la web de Genbeta Dev, distribuidas en 10 hilos. Precisamente esta capacidad de concurrencia permitirá comprobar condiciones de carrera o bloqueos, ya que el comportamiento de las peticiones es más natural que si se realizan las 100 seguidas de un bucle.

En el informe que proporciona la herramienta se puede observar el mínimo, máximo, la media y la moda de las mediciones, así como posibles errores o el total de datos descargados.

ApacheBench es una herramienta de software libre y se distribuye bajo los términos de la Licencia de Apache.

Capítulo III

3. Entorno de pruebas y diseño del estudio

A priori, de los distintos tipos de benchmarking que se ha visto anteriormente en esta investigación, se puede predecir que el Benchmarking interno es la que va a conseguir los resultados del mejor rendimiento para su implementación, en comparación con los demás tipos de benchmarking.

Con este tipo de benchmarking se va a medir el rendimiento de los componentes hardware del ordenador, llevando el componente seleccionado a su máxima capacidad. Se realizará el análisis a bajo nivel o sea que mide directamente el rendimiento de los componentes: uso de CPU, uso de RAM y tiempo de respuesta a peticiones al servidor.

En la investigación se ha planteado realizar una serie de pruebas de rendimiento a cada uno de los servidores de una manera lo más fiable posible. Se va a estudiar la manera de como dejar el sistema en perfecto funcionamiento para la posterior realización de las pruebas de rendimiento, para lo cual se va a realizar algunas configuraciones iniciales a cada uno de

los servidores para la ejecución del Módulo de Catálogos para el sistema de Descripción y Valoración del Departamento de Recursos Humanos de la Escuela Superior Politécnica de Chimborazo.

3.1. Configuración del sistema

Para la realización del presente estudio, se ha definido un escenario común al cuál se le han aplicado ciertas modificaciones para poder medir el rendimiento. En concreto se ha respetado al máximo posible la similitud entre los casos de estudio y la configuración de la máquina física. A continuación se va a detallar cuales son las configuraciones hardware y versiones software utilizadas.

3.1.1. Configuración Hardware

El sistema utilizado para realizar las pruebas es una portátil HP 6510B que dispone de un procesador Dual Core a 1.8 GHz. La memoria principal del sistema empleado son 2 GB de RAM. El disco duro es un Samsung Serial ATA que dispone de 320 GB. El sistema de red está configurado con un controlador Pci Gigabit Ethernet Broadcom NetLink (NIC 10/100/100), módem v9.2 de 56K.

Tabla III: *Especificación hardware para el servidor de prueba*

Hardware	Modelo
Procesador	Intel Core 2 Duo de 1.8 Ghz (cache L2 de 2MB)
Chipset	Mobile Intel GM965
Memoria	Memoria DDR2 SDRAM de 667 MHz
Unidad de Disco Duro	SATA de 320 GB de 5400 rpm, HP 3D DriveGuard
Medios Extraibles	Unidad óptica de 12,7mm: DVD+/-RW SuperMulti DL LightScribe

Pantalla	WXGA diagonal de 14,1 pulgadas (1280 x 800) y 16M de colores
Gráficos	Intel GMA X3100, hasta 348 MB de memoria de sistema compartido
Audio	Audio de Alta Definición, altavoces estéreo
Soporte Inalámbrico	Intel PRO/Wireless 802.11a/b/g Bluetooth 2.0
Comunicaciones	Controlador PCI Gigabit Ethernet Broadcom NetLink (NIC 10/100/1000) Modem v9.2 de 56K
Puertos	4 puertos USB 2.0 RJ-11/módem RJ-45/ethernet Salida S-video TV
Dispositivo de Entrada	Teclado de tamaño integral Latino Touchpad con zona de desplazamiento Sensor de Huella Digital HP

3.1.2. Configuración Software

El sistema operativo que se ha elegido para utilizar como host en ambos casos del estudio, ha sido Centos 5.

Junto con un conjunto completo de la plataforma Java, Edición Empresarial en su versión 6; para la ejecución de los servidores como son GlassFish y JBoss Application Server

Un conjunto de herramientas benchmarking del conjunto de sysstat. Esto se detalla en detalle en Pruebas Diseñadas.

Tabla IV: *Especificación software para la configuración del servidor*

Software	**Versión**
Sistema Operativo	Centos 5
Java Runtime Enviroment	jre 7 update 25, de Oracle Corporation
Java Development Kit	jdk 7 update 25, de Oracle Corporation
Servidor GlassFish	3.1.2.2 Full Profile

Servidor JBoss Application Server	7.1.1 Full Profile
Herramientas de Benchmarking	• syststat

3.2. Pruebas Diseñadas

Estas pruebas están diseñado para la demostración de la hipótesis y llegar a una conclusión sobre está.

3.2.1. Paquete sysstat[20]

El nombre sar proviene de las siglas “system activity report” (Informe de actividad del sistema). En Linux se encuentra normalmente en el paquete sysstat. El paquete sysstat incluye programas y scripts para recopilar y mostrar información sobre el rendimiento del sistema, generado informes detallados. Este conjunto de programas puede resultar de mucha utilidad a la hora de detectar cuellos de botellas y para hacernos una idea de cómo se utiliza el sistema a lo largo del día.

El programa que se ocupa de recopilar la información se llama sadc “system activity data collector” (Recolector de datos de la actividad del sistema). Obtiene su información principal del kernel a través del sistema de archivos virtual en /proc.

Como demostró el Dr. Heisenberg, el simple hecho de realizar una medida cambia los datos medidos. Cualquier herramienta dedicada a recopilar datos sobre rendimiento tiene un cierto impacto negativo en el rendimiento del sistema, pero con sar, este parece ser mínimo.

20 **SUPERVISION DE ACTIVIDADES DEL SISTEMA (SAR)**
http://docs.oracle.com/cd/E24842_01/html/E23086/spmonitor-8.html
2013-08-01

El comando sar puede realizar las siguientes tareas:

- Organizar y ver datos sobre la actividad del sistema.
- Acceder a los datos de actividad del sistema con una solicitud especial.
- Generar informes automáticos para medir y supervisar el rendimiento del sistema e informes de solicitud especial para identificar problemas específicos de rendimientos.

La opción interval para todos los comandos sar sera 2 000 000 ya que mayor sea las peticiones mejor resultados se tendrá al momento de recoger los datos obtenidos por el servidor mientras que count van hacer la concurrencia con la que se va ha trabajar. Esto se detalla en el sub-capítulo Población.

3.2.1.1. Uso de RAM

Se utiliza el comando sar -r para informar sobre las siguientes actividades del asignador de memoria de núcleo (KMA).

sar -r [interval [count]]

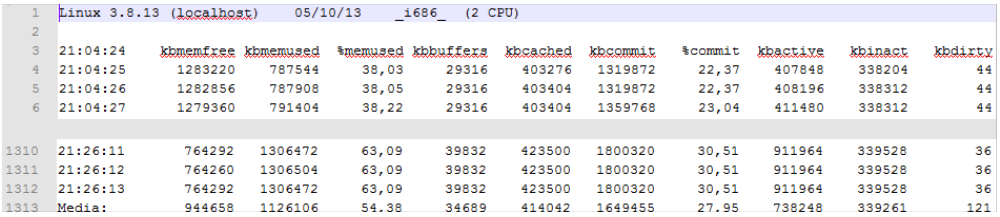
Tabla V: *Resultados que muestra el comando sar -r*

Valores	Descripción
kbmemfree	Cantidad de memoria libre disponible en kilobytes
kbmemused	Cantidad de memoria usada in kilobytes
%memused	Porcentaje de memoria usada
kbbuffers	Cantidad de memoria usada como buffers por el kernel en kilobytes
kbcached	Cantidad de memoria usada en el cache de datos del kernel in kilobytes

kbcommit	Cantidad de memoria en kilobytes que necesita para la actual carga de trabajo. Esto es un estimado de como mucho RAM/swap esto es necesario para garantizar que nunca se quede sin memoria
%commit	Porcentaje de memoria que necesita para la carga de trabajo en relación de la cantidad total de la memoria RAM+swap
kbactive	Cantidad de memoria activa en kilobytes
kbinact	Cantidad de memoria inactiva en kilobytes

A continuación se muestra el resultado final de la prueba de benchmarking aplicado al servidor GlassFish y JBoss AS.

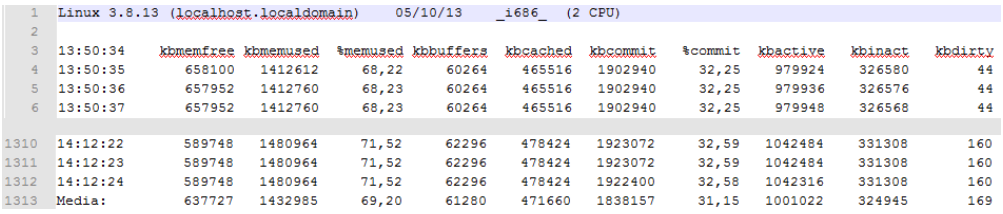
Las resultados que se llevo a cabo es cuando la concurrencia es igual a 9 personas.



```
Linux 3.8.13 (localhost)     05/10/13     _i686_   (2 CPU)

21:04:24    kbmemfree kbmemused  %memused kbbuffers  kbcached  kbcommit   %commit  kbactive   kbinact   kbdirty
21:04:25      1283220    787544     38,03     29316    403276   1319872     22,37    407848    338204        44
21:04:26      1282856    787908     38,05     29316    403404   1319872     22,37    408196    338312        44
21:04:27      1279360    791404     38,22     29316    403404   1359768     23,04    411480    338312        44

21:26:11       764292   1306472     63,09     39832    423500   1800320     30,51    911964    339528        36
21:26:12       764260   1306504     63,09     39832    423500   1800320     30,51    911964    339528        36
21:26:13       764292   1306472     63,09     39832    423500   1800320     30,51    911964    339528        36
Media:         944658   1126106     54,38     34689    414042   1649455     27,95    738248    339261       121
```

Figura 24: *Resultados RAM, servidor GlassFish con una concurrencia igual a 9*

Realizado por: Diego P. Tamayo Barriga

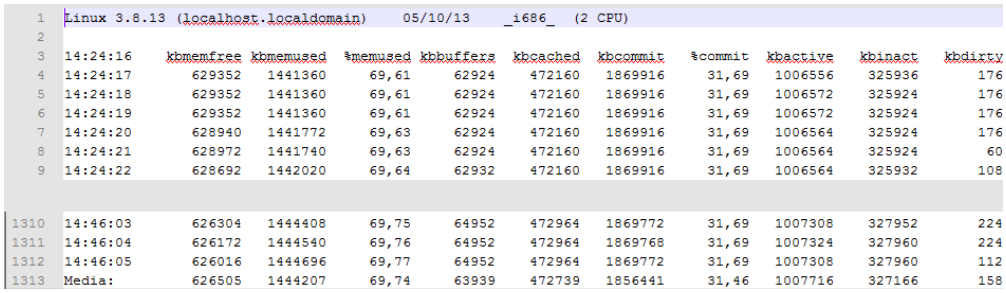


```
Linux 3.8.13 (localhost.localdomain)     05/10/13     _i686_   (2 CPU)

13:50:34    kbmemfree kbmemused  %memused kbbuffers  kbcached  kbcommit   %commit  kbactive   kbinact   kbdirty
13:50:35       658100   1412612     68,22     60264    465516   1902940     32,25    979924    326580        44
13:50:36       657952   1412760     68,23     60264    465516   1902940     32,25    979936    326576        44
13:50:37       657952   1412760     68,23     60264    465516   1902940     32,25    979948    326568        44

14:12:22       589748   1480964     71,52     62296    478424   1923072     32,59   1042484    331308       160
14:12:23       589748   1480964     71,52     62296    478424   1923072     32,59   1042484    331308       160
14:12:24       589748   1480964     71,52     62296    478424   1922400     32,58   1042316    331308       160
Media:         637727   1432985     69,20     61280    471660   1838157     31,15   1001022    324945       169
```

Figura 25: *Resultados RAM, servidor JBoss AS con una concurrencia igual a 9*

Realizado por: Diego P. Tamayo Barriga

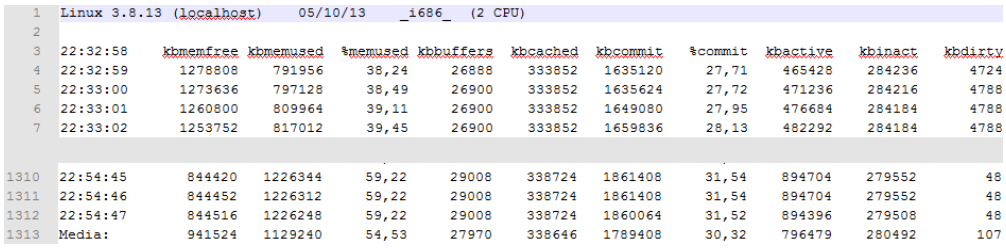
Las resultados que se llevo a cabo es cuando la concurrencia es igual a 200 personas.



```
Linux 3.8.13 (localhost.localdomain)    05/10/13    _i686_   (2 CPU)

14:24:16    kbmemfree kbmemused  %memused kbbuffers  kbcached  kbcommit   %commit  kbactive   kbinact   kbdirty
14:24:17       629352   1441360     69,61     62924    472160   1869916     31,69   1006556    325936       176
14:24:18       629352   1441360     69,61     62924    472160   1869916     31,69   1006572    325924       176
14:24:19       629352   1441360     69,61     62924    472160   1869916     31,69   1006572    325924       176
14:24:20       628940   1441772     69,63     62924    472160   1869916     31,69   1006564    325924       176
14:24:21       628972   1441740     69,63     62924    472160   1869916     31,69   1006564    325924        60
14:24:22       628692   1442020     69,64     62932    472160   1869916     31,69   1006564    325932       108

14:46:03       626304   1444408     69,75     64952    472964   1869772     31,69   1007308    327952       224
14:46:04       626172   1444540     69,76     64952    472964   1869768     31,69   1007324    327960       224
14:46:05       626016   1444696     69,77     64952    472964   1869772     31,69   1007308    327960       112
Media:         626505   1444207     69,74     63939    472739   1856441     31,46   1007716    327166       158
```

Figura 26: Resultados RAM, servidor GlassFish con una concurrencia igual a 200

Realizado por: Diego P. Tamayo Barriga



```
Linux 3.8.13 (localhost)    05/10/13    _i686_   (2 CPU)

22:32:58    kbmemfree kbmemused  %memused kbbuffers  kbcached  kbcommit   %commit  kbactive   kbinact   kbdirty
22:32:59      1278808    791956     38,24     26888    333852   1635120     27,71    465428    284236      4724
22:33:00      1273636    797128     38,49     26900    333852   1635624     27,72    471236    284216      4788
22:33:01      1260800    809964     39,11     26900    333852   1649080     27,95    476684    284184      4788
22:33:02      1253752    817012     39,45     26900    333852   1659836     28,13    482292    284184      4788

22:54:45       844420   1226344     59,22     29008    338724   1861408     31,54    894704    279552        48
22:54:46       844452   1226312     59,22     29008    338724   1861408     31,54    894704    279552        48
22:54:47       844516   1226248     59,22     29008    338724   1860064     31,52    894396    279508        48
Media:         941524   1129240     54,53     27970    338646   1789408     30,32    796479    280492       107
```

Figura 27: *Resultados RAM, servidor JBoss AS con una concurrencia igual a 200*

Realizado por: Diego P. Tamayo Barriga

3.2.1.2. Uso de CPU

Se utiliza el comando sar -u para visualizar las estadísticas de uso de la CPU. El comando sar sin opciones es equivalente al comando sar -u. En algún momento, el procesador estará ocupado o inactivo. Cuando el procesador está ocupado, se encuentra en modo de usuario o en modo de sistema. Cuando el procesador está inactivo, está esperando la finalización de E/S o está paralizado sin trabajo que hacer.

sar -u [ALL] [interval [count]]

sar [ALL] [interval [count]]

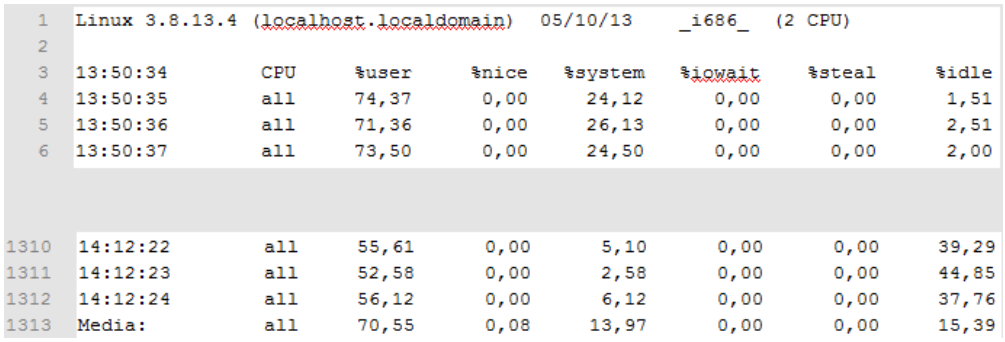
La opción ALL indica que muestre todos los campos del CPU.

Tabla VI: *Resultados que muestra el comando sar -u*

Valores	Descripción
%user	Porcentaje del CPU utilizado mientras ocurre la ejecución del nivel de usuario (aplicación).
%usr	Porcentaje de CPU utilizado mientras ocurre la ejecución del nivel de usuario (aplicación)
%nice	Porcentaje del CPU utilizado mientras ocurre la ejecución del nivel de usuario con prioridad
%system	Porcentaje del CPU utilizado mientras ocurre el nivel del sistema (kernel)
%sys	Porcentaje del CPU utilizado mientras ocurre el nivel del sistema (kernel)
%iowait	Porcentaje de tiempo que el CPU o CPUs está inactivo en un proceso de requerimiento de E/S del disco duro
%steal	Porcentaje de tiempo de espera involuntaria del CPU o CPUs virtuales mientras el hipervisor estaba sirviendo otro proceso virtual.
%idle	Muestra el porcenaje de tiempo durante el cual el procesador esta incativo y no en espera de la finalización de E/S

A continuación se muestra el resultado final de la prueba de benchmarking aplicado al servidor GlassFish y JBoss AS.

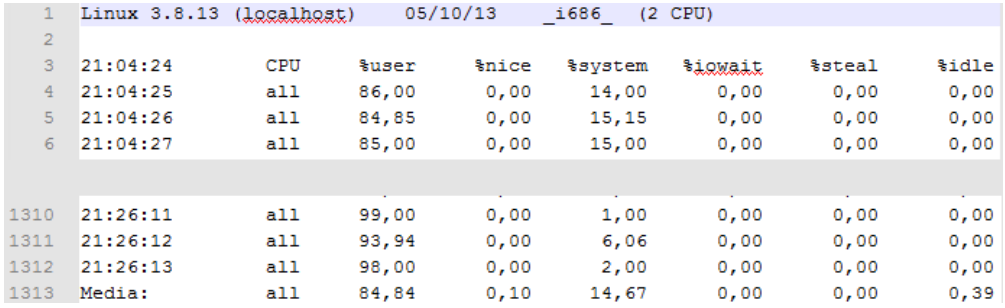
Las resultados que se llevo a cabo es cuando la concurrencia es igual a 9 personas.



1	Linux 3.8.13.4 (localhost.localdomain)	05/10/13	_i686_	(2 CPU)			
2							
3	13:50:34	CPU	%user	%nice	%system	%iowait	%steal	%idle
4	13:50:35	all	74,37	0,00	24,12	0,00	0,00	1,51
5	13:50:36	all	71,36	0,00	26,13	0,00	0,00	2,51
6	13:50:37	all	73,50	0,00	24,50	0,00	0,00	2,00
1310	14:12:22	all	55,61	0,00	5,10	0,00	0,00	39,29
1311	14:12:23	all	52,58	0,00	2,58	0,00	0,00	44,85
1312	14:12:24	all	56,12	0,00	6,12	0,00	0,00	37,76
1313	Media:	all	70,55	0,08	13,97	0,00	0,00	15,39

Figura 28: *Resultados CPU, servidor GlassFish con una concurrencia igual a 9*

Realizado por: Diego P. Tamayo Barriga

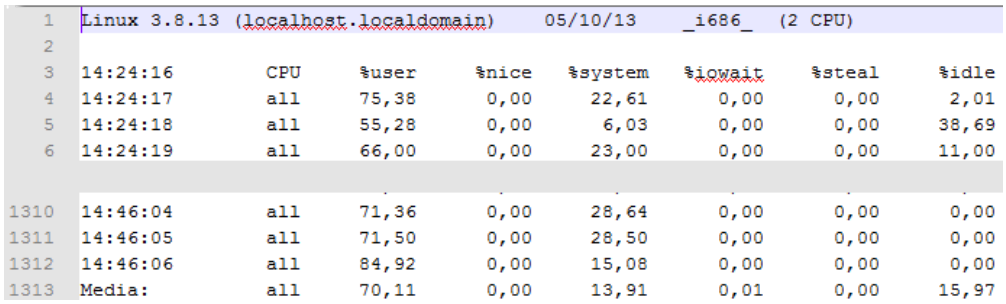


1	Linux 3.8.13 (localhost)	05/10/13	_i686_	(2 CPU)			
2							
3	21:04:24	CPU	%user	%nice	%system	%iowait	%steal	%idle
4	21:04:25	all	86,00	0,00	14,00	0,00	0,00	0,00
5	21:04:26	all	84,85	0,00	15,15	0,00	0,00	0,00
6	21:04:27	all	85,00	0,00	15,00	0,00	0,00	0,00
1310	21:26:11	all	99,00	0,00	1,00	0,00	0,00	0,00
1311	21:26:12	all	93,94	0,00	6,06	0,00	0,00	0,00
1312	21:26:13	all	98,00	0,00	2,00	0,00	0,00	0,00
1313	Media:	all	84,84	0,10	14,67	0,00	0,00	0,39

Figura 29: *Resultados CPU, servidor JBoss AS con una concurrencia igual a 9*

Realizado por: Diego P. Tamayo Barriga

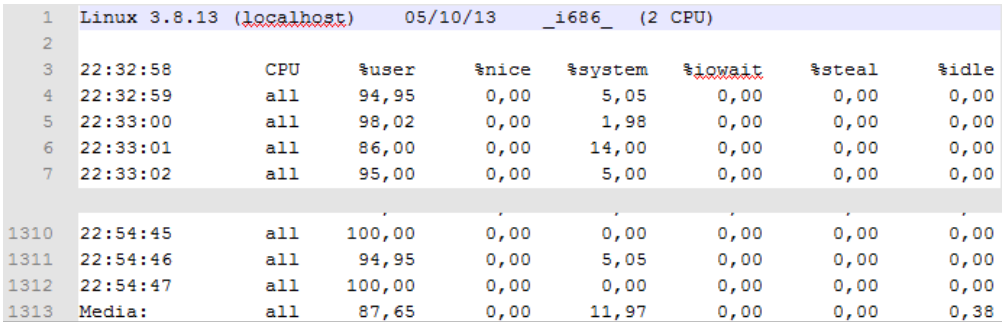
Las resultados que se llevo a cabo es cuando la concurrencia es igual a 200 personas.



1	Linux 3.8.13 (localhost.localdomain)	05/10/13	_i686_	(2 CPU)			
2							
3	14:24:16	CPU	%user	%nice	%system	%iowait	%steal	%idle
4	14:24:17	all	75,38	0,00	22,61	0,00	0,00	2,01
5	14:24:18	all	55,28	0,00	6,03	0,00	0,00	38,69
6	14:24:19	all	66,00	0,00	23,00	0,00	0,00	11,00
1310	14:46:04	all	71,36	0,00	28,64	0,00	0,00	0,00
1311	14:46:05	all	71,50	0,00	28,50	0,00	0,00	0,00
1312	14:46:06	all	84,92	0,00	15,08	0,00	0,00	0,00
1313	Media:	all	70,11	0,00	13,91	0,01	0,00	15,97

Figura 30: *Resultados CPU, servidor GlassFish con una concurrencia igual a 200*

Realizado por: Diego P. Tamayo Barriga



```
Linux 3.8.13 (localhost)    05/10/13    _i686_   (2 CPU)

22:32:58        CPU     %user     %nice   %system   %iowait    %steal     %idle
22:32:59        all     94,95      0,00      5,05      0,00      0,00      0,00
22:33:00        all     98,02      0,00      1,98      0,00      0,00      0,00
22:33:01        all     86,00      0,00     14,00      0,00      0,00      0,00
22:33:02        all     95,00      0,00      5,00      0,00      0,00      0,00

1310    22:54:45        all    100,00      0,00      0,00      0,00      0,00      0,00
1311    22:54:46        all     94,95      0,00      5,05      0,00      0,00      0,00
1312    22:54:47        all    100,00      0,00      0,00      0,00      0,00      0,00
1313    Media:          all     87,65      0,00     11,97      0,00      0,00      0,38
```

Figura 31: *Resultados CPU, servidor JBoss AS con una concurrencia igual a 200*

Realizado por: Diego P. Tamayo Barriga

3.2.1.3. Uso de la Red

Se utiliza el comando sar -n para visualizar las estadísticas de uso de la red.

sar -n {keyword | ALL} [interval [count]]

Con keyword igual a DEV, reporta estadisticas desde los dispositivos de red.

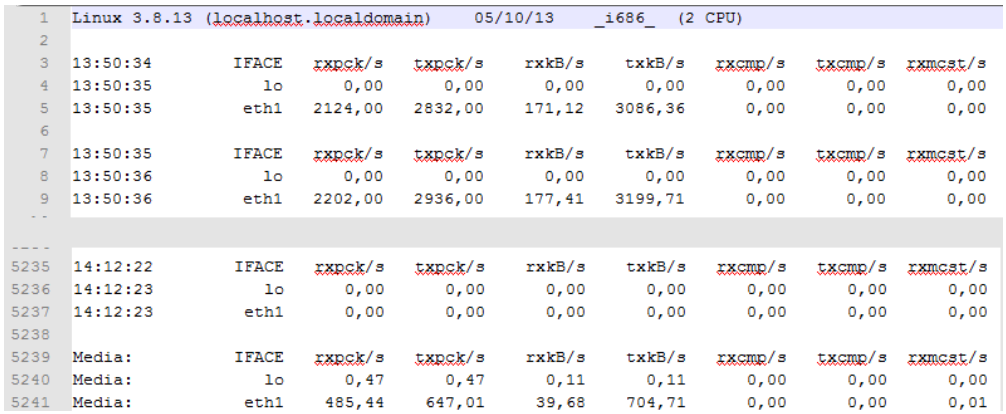
Los siguientes valores son desplegados.

Tabla VII: *Resultados que muestra el comando sar -n*

Valores	Descripción
IFACE	Nombre de la interfaz de red de la cual reporta las estadísticas
rxpck/s	Número total de paquetes recibidos por segundo
txpck/s	Número total de paquetes transmitidos por segundo
rkB/s	Número total de kilobytes recibidos por segundo
txkB/s	Número total de kilobytes transmitidos por segundo
rxcmp/s	Número de paquetes comprimidos recibidos por segundo
txcmp/s	Número de paquetes comprimidos transmitidos por segundo
rxmcst/s	Número de paquetes multicast recibidos por segundo

A continuación se muestra el resultado final de la prueba de benchmarking aplicado al servidor GlassFish y JBoss AS.

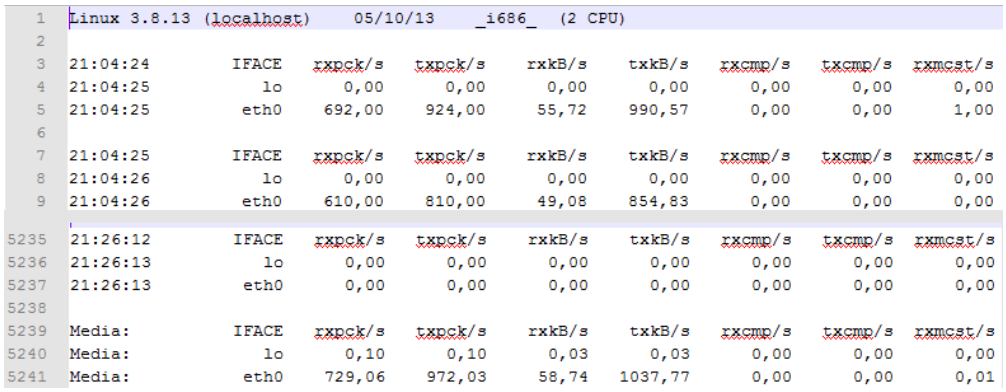
Las resultados que se llevo a cabo es cuando la concurrencia es igual a 9 personas.



```
Linux 3.8.13 (localhost.localdomain)    05/10/13    _i686_   (2 CPU)

13:50:34        IFACE   rxpck/s   txpck/s   rxkB/s   txkB/s   rxcmp/s   txcmp/s  rxmcst/s
13:50:35           lo      0,00      0,00     0,00     0,00      0,00      0,00      0,00
13:50:35         eth1   2124,00   2832,00   171,12  3086,36      0,00      0,00      0,00

13:50:35        IFACE   rxpck/s   txpck/s   rxkB/s   txkB/s   rxcmp/s   txcmp/s  rxmcst/s
13:50:36           lo      0,00      0,00     0,00     0,00      0,00      0,00      0,00
13:50:36         eth1   2202,00   2936,00   177,41  3199,71      0,00      0,00      0,00

14:12:22        IFACE   rxpck/s   txpck/s   rxkB/s   txkB/s   rxcmp/s   txcmp/s  rxmcst/s
14:12:23           lo      0,00      0,00     0,00     0,00      0,00      0,00      0,00
14:12:23         eth1      0,00      0,00     0,00     0,00      0,00      0,00      0,00

Media:          IFACE   rxpck/s   txpck/s   rxkB/s   txkB/s   rxcmp/s   txcmp/s  rxmcst/s
Media:             lo      0,47      0,47     0,11     0,11      0,00      0,00      0,00
Media:           eth1    485,44    647,01    39,68   704,71      0,00      0,00      0,01
```

Figura 32: *Resultados RED, servidor GlassFish con una concurrencia igual a 9*

Realizado por: Diego P. Tamayo Barriga



```
Linux 3.8.13 (localhost)    05/10/13    _i686_   (2 CPU)

21:04:24        IFACE   rxpck/s   txpck/s   rxkB/s   txkB/s   rxcmp/s   txcmp/s  rxmcst/s
21:04:25           lo      0,00      0,00     0,00     0,00      0,00      0,00      0,00
21:04:25         eth0    692,00    924,00    55,72   990,57      0,00      0,00      1,00

21:04:25        IFACE   rxpck/s   txpck/s   rxkB/s   txkB/s   rxcmp/s   txcmp/s  rxmcst/s
21:04:26           lo      0,00      0,00     0,00     0,00      0,00      0,00      0,00
21:04:26         eth0    610,00    810,00    49,08   854,83      0,00      0,00      0,00

21:26:12        IFACE   rxpck/s   txpck/s   rxkB/s   txkB/s   rxcmp/s   txcmp/s  rxmcst/s
21:26:13           lo      0,00      0,00     0,00     0,00      0,00      0,00      0,00
21:26:13         eth0      0,00      0,00     0,00     0,00      0,00      0,00      0,00

Media:          IFACE   rxpck/s   txpck/s   rxkB/s   txkB/s   rxcmp/s   txcmp/s  rxmcst/s
Media:             lo      0,10      0,10     0,03     0,03      0,00      0,00      0,00
Media:           eth0    729,06    972,03    58,74  1037,77      0,00      0,00      0,01
```

Figura 33: *Resultados RED, servidor JBoss AS con una concurrencia igual a 9*

Realizado por: Diego P. Tamayo Barriga

Las resultados que se llevo a cabo es cuando la concurrencia es igual a 200 personas.

```
1    Linux 3.8.13 (localhost.localdomain)     05/10/13    _i686_   (2 CPU)
2
3    14:24:16        IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
4    14:24:17           lo      0,00      0,00      0,00      0,00      0,00      0,00      0,00
5    14:24:17         eth1   2156,00   2868,00    173,86   3105,12      0,00      0,00      0,00
6
7    14:24:17        IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
8    14:24:18           lo      0,00      0,00      0,00      0,00      0,00      0,00      0,00
9    14:24:18         eth1    410,00    552,00     32,90    617,96      0,00      0,00      0,00

5235 14:46:05        IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
5236 14:46:06           lo      6,00      6,00      1,75      1,75      0,00      0,00      0,00
5237 14:46:06         eth1     72,00     96,00      5,80    104,62      0,00      0,00      0,00
5238
5239 Media:          IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
5240 Media:             lo      0,47      0,47      0,11      0,11      0,00      0,00      0,00
5241 Media:           eth1    472,61    629,51     38,09    682,90      0,00      0,00      0,01
```

Figura 34: *Resultados RED, servidor GlassFish con una concurrencia igual a 200*

Realizado por: Diego P. Tamayo Barriga

```
1    Linux 3.8.13 (localhost)     05/10/13    _i686_   (2 CPU)
2
3    22:32:58        IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
4    22:32:59           lo      0,00      0,00      0,00      0,00      0,00      0,00      0,00
5    22:32:59         eth0      0,00      0,00      0,00      0,00      0,00      0,00      0,00
6
7    22:32:59        IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
8    22:33:00           lo      0,00      0,00      0,00      0,00      0,00      0,00      0,00
9    22:33:00         eth0      0,00      0,00      0,00      0,00      0,00      0,00      0,00

5231 22:54:45        IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
5232 22:54:46           lo      0,00      0,00      0,00      0,00      0,00      0,00      0,00
5233 22:54:46         eth0    404,00    536,00     32,61    564,28      0,00      0,00      0,00
5234
5235 22:54:46        IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
5236 22:54:47           lo      0,00      0,00      0,00      0,00      0,00      0,00      0,00
5237 22:54:47         eth0      0,00      0,00      0,00      0,00      0,00      0,00      0,00
5238
5239 Media:          IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
5240 Media:             lo      0,10      0,10      0,03      0,03      0,00      0,00      0,00
5241 Media:           eth0    692,79    909,50     55,85    967,21      0,00      0,00      0,01
```

Figura 35: *Resultados RED, servidor GlassFish con una concurrencia igual a 200*

Realizado por: Diego P. Tamayo Barriga

En las figuras anteriores se muestra una parte del resultado obtenido en la prueba de medición con la herramienta sar para ver toda la información se vera detallado en el CD[21] que viene con este informe, en el encontrarán

21 **DATOS MUESTRALES.,** CDs., TAMAYO D., ESPOCH., 2013-10-22

todos los archivos generados.

3.3. Determinando parámetros de comparación

En está sección se implementa un análisis estadístico comparativo entre los servidores de aplicación GlassFish y JBoss AS.

Para la comprobación de la hipótesis se ha decidido utilizar una técnica estadística, misma que ha permitido la obtención de resultados de cada uno de los componentes hardware que han sido estudiado y expresado en valores numéricos, permitiendo elegir la mejor opción para su posterior implementación.

3.3.1. Definición de los parámetros a comparar

Como se comento en este capítulo en la parte de Pruebas Diseñadas los parámetros para la comparación van a ser:

- Uso de CPU
- Uso de RAM
- Uso de RED

Para la elección del mejor servidor de aplicaciones para la Plataforma Java EE se debe tener en cuenta el que menor consumo que tenga en cada uno de los parámetros, para su posterior implementación del Módulo de Catálogos del Sistema de Recursos Humanos.

A continuación en la siguiente tabla, se detalla los parámetros y sus valores a usar dentro de la presente investigación.

Tabla VIII: *Valores de los parámetros a usar*

Parámetro	Valor Porcentual
Uso de CPU	50%
Uso de RAM	40%
Uso de RED	10%
Total	100%

Los valores que se le da a cada parámetro para la puntuación global está tomada de acuerdo a la importancia que tiene cada componente dentro del servidor, siendo el CPU el que tiene un valor alto ya que no cuenta con un procesador aceptable para la ejecución, frente a la RAM con la que cuenta con una cantidad aceptable y la RED siendo esta no tan importante en el estudio por que el módulo va implementarse dentro de la Intranet de la Escuela Superior Politécnica de Chimborazo.

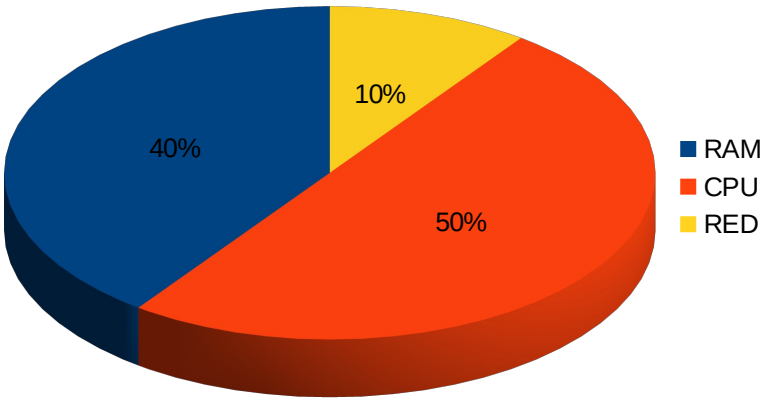


Figura 36: *Valores porcentuales de los parámetros*

Realizado por: Diego P. Tamayo Barriga

Una vez que se a definido los valores de los parámetros para cada componente hardware de acuerdo a la importancia dentro del servidor, se

procederá al análisis comparativo de cada uno de estos para lo cual se hará uso de la Estadística Inferencial.

3.3.2. Métodos, técnicas y procedimientos

Para la realización del presente proyecto previamente han sido definidos un conjunto de métodos, técnicas y procedimientos que son la base para la ejecución de la investigación.

Los elementos mencionados se detallan a continuación:

Método

Está investigación tiene como marco principal el método científico experimental; de forma general, este método:

- Plantea el problema
- Describe una hipótesis
- Recopila información suficiente para comprobar la hipótesis con el uso de experimentos.
- Analiza y difunde los resultados.

Técnicas y procedimientos

Para el desarrollo de las siguientes secciones de este capítulo, es necesario recordar la hipótesis planteada previamente a la ejecución de la tesis.

"El servidor de aplicaciones GlassFish es más eficiente en el consumo de recursos en relación al servidor de aplicaciones Jboss."

Para la comprobación de está hipótesis, se especifica la población y la muestra que serán el objeto de estudio. Así mismo se utilizará la

estadística descriptiva e inferencial para alcanzar el objetivo[3]

Población

No se puede establecer la población total ya que no se cuenta con un período de uso del Módulos de Catálogos del Sistema de Recursos Humanos.

1. Pero en la actualidad se sabe que concurrentemente las personas que utilizarán el módulo son:
    - Departamento de Recursos Humanos: 4 personas.
    - Departamento Financiero: 3 personas.
    - Rectorado: 2 personas
2. En una aproximación en el futuro se va ha tener que el sistema ya no solo va ha ser usado por los tres departamento anteriormente mencionados. Se ha considerando una concurrencia de todos los departamentos de la Escuela Superior Politécnica de Chimborazo y se tiene que aproximadamente unas 200 personas pueden estar haciendo peticiones concurrentemente al servidor.

En los dos caso anteriores se tiene que demostrar las hipótesis con una población infinita de resultados.

Para la decisión final se tomará en cuenta la concurrencia a los servidores de aplicación. Conociendo que la concurrencia actualidad es la más importante se le dará un peso del 70% en relación a la concurrencia futura que se le da un peso del 30% al momento de la decisión final.

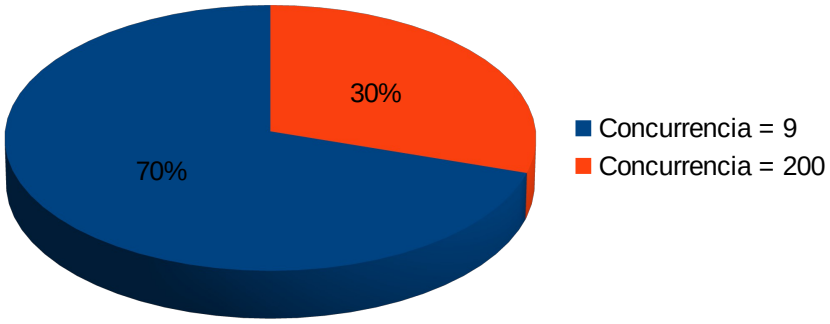


Figura 37: *Parámetro de concurrencia para la decisión final*

Realizado por: Diego P. Tamayo Barriga

Método de muestreo

Para la demostración de la hipótesis de está investigación se ha seleccionado una muestra a través del método probabilístico. Para el cálculo de la muestra se trabajó con el nivel de confianza del 97%.

Muestra

Para realizar el estudio comparativo estadístico del servidor de aplicaciones más eficiente en el consumo de los recursos se han realizado dos pruebas.

1. La primera con una concurrencia de 9 personas que es como esta en la actualidad.
2. La segunda con una proyección a un crecimiento para las distintos Unidades o Departamentos que existen en la Escuela Superior Politécnica de Chimborazo. Entre las cuales se puede decir que la concurrencia al sistema sería aproximada de unas 200 personas.

Para realizar el análisis, se procede a hacer los cálculos en el Apéndice A

3.4. Comparación entre GlassFish y JBoss AS

Para la demostración se utilizará la diferencia de dos medias muestrales para todos los conjuntos obtenidos de cada servidor.

Para la demostración de la hipótesis se utilizará la distribución que más se acople con nuestros parámetros, entre las distribuciones que se tiene para la demostración se listan en la siguiente figura:

	Cuando se conoce la desviación estándar de la población	Cuando no se conoce la desviación estándar de la población
El tamaño de muestra n>30	distribución normal, tabla z	distribución normal, tabla z
El tamaño de muestra n<30 y suponemos que la población es normal o aproximadamente normal.	distribución normal, tabla z	distribución t, tabla t

Figura 38: *Distribución de Hipótesis*

Realizado por: Diego P. Tamayo Barriga

Se utilizará la Prueba de 5 pasos para la demostración de hipótesis que se detalla en el Anexo B

3.4.1. Prueba de 5 pasos para la demostración de la hipótesis

Por comodidad para el lector μ es las variables para la demostración de la hipótesis. Para referirnos a la muestra se utilizará n

3.4.1.1. Uso de CPU para una concurrencia igual a 9

$\mu_1 = El\ uso\ de\ CPU\ del\ servidor\ de\ aplicaciones\ Glassfish$

$\mu_2 = El\ uso\ de\ CPU\ del\ servidor\ de\ aplicaciones\ JBoss\ AS$

Paso 1 – Tipo de prueba, establecer hipótesis y nivel de confianza

Hipótesis nula: El uso del CPU del servidor de aplicaciones GlassFish es

igual al uso del CPU del servidor de aplicaciones JBoss AS

$$H_0: \mu_1 = \mu_2$$

Hipótesis alternativa: El uso del CPU del servidor de aplicaciones GlassFish debe ser menor que el uso del CPU del servidor de aplicaciones JBoss AS, para decir que es más eficiente en el consumo de la CPU

$$H_a: \mu_1 < \mu_2 \Rightarrow Prueba\ de\ cola\ izquierda$$

Nivel de confianza

$$\alpha = 0.03$$

Paso 2 – Seleccionar distribución de probabilidad y valor crítico

Distribución normal por que $n > 30$ y se desconoce la desviación estándar de la población

El valor crítico es $z = -2.17$

Paso 3 – Calcular el error estándar y estandarizar las medias de las muestras

Como no se conoce la desviación estándar de la población se va ha decir que la desviación estándar de la muestra es igual al de la población.

$$\sigma = S$$

Calcular el error estándar

$$\sigma_{\bar{x}_1 - \bar{x}_2} = \sqrt{\frac{\sigma_1^2}{n} + \frac{\sigma_2^2}{n}} = 0.43379$$

Calcular z estandarizado para las medias de las muestras

$$z=\frac{(\bar{x}_1-\bar{x}_2)-(\bar{\mu}_1-\bar{\mu}_2)H_0}{\sigma_{\bar{x}_1-\bar{x}_2}}$$

$$(\bar{\mu}_1-\bar{\mu}_2)H_0=0$$

$$z=-33.076$$

Paso 4 – Gráfica de la distribución

Figura 39: *Gráfica de la distribución de la normal*

Realizado por: Diego P. Tamayo Barriga

Paso 5 – Comprobar estadístico muestral con valores críticos

Se comprueba que $-33.067<-2.17$ y cae fuera de la zona donde se aceptación la hipótesis nula H_0

Por último se rechaza la hipótesis nula con un ERROR TIPO I con una probabilidad de cometer igual a α

Entonces se puede concluir que: El servidor de aplicaciones GlassFish es más eficiente en el uso del CPU en relación al servidor de aplicaciones

JBoss AS

3.4.1.2. Uso de RAM para una concurrencia igual a 9

$\mu_1 = El\ uso\ de\ RAM\ \ del\ servidor\ de\ aplicaciones\ Glassfish$

$\mu_2 = El\ uso\ de\ RAM\ \ del\ servidor\ de\ aplicaciones\ JBoss\ AS$

Paso 1 – Tipo de prueba, establecer hipótesis y nivel de confianza

Hipótesis nula: El uso de la RAM del servidor de aplicaciones GlassFish es igual al uso del RAM del servidor de aplicaciones JBoss AS

$H_0 : \mu_1 = \mu_2$

Hipótesis alternativa: El uso de la RAM del servidor de aplicaciones GlassFish debe ser menor que el uso de la RAM del servidor de aplicaciones JBoss AS, para decir que es más eficiente en el consumo de la RAM

$H_a : \mu_1 < \mu_2 \Rightarrow Prueba\ de\ cola\ izquierda$

Nivel de confianza

$\alpha = 0.03$

Paso 2 – Seleccionar distribución de probabilidad y valor crítico

Distribución normal por que $n > 30$ y se desconoce la desviación estándar de la población

El valor crítico es $z = -2.17$

Paso 3 – Calcular el error estándar y estandarizar las medias de las muestras

Como no se conoce la desviación estándar de la población se va ha decir que la desviación estándar de la muestra es igual a la de la población.

$$\sigma = S$$

Calcular el error estándar

$$\sigma_{\bar{x}_1 - \bar{x}_2} = \sqrt{\frac{\sigma_1^2}{n} + \frac{\sigma_2^2}{n}} = 0.2474$$

Calcular z estandarizado para las medias de las muestras

$$z = \frac{(\bar{x}_1 - \bar{x}_2) - (\bar{\mu}_1 - \bar{\mu}_2) H_0}{\sigma_{\bar{x}_1 - \bar{x}_2}}$$

$$(\bar{\mu}_1 - \bar{\mu}_2) H_0 = 0$$

$$z = -59{,}9118$$

Paso 4 – Gráfica de la distribución

Se toma en referencia la Figura 39: Gráfica de la distribución de la normal

Paso 5 – Comprobar estadístico muestral con valores críticos

Se comprueba que $-59{,}9118 < -2.17$ y cae fuera de la zona donde se aceptación la hipótesis nula H_0

Por último se rechaza la hipótesis nula con un ERROR TIPO I con una probabilidad de cometer igual a α

Se puede concluir que: El servidor de aplicaciones GlassFish es más eficiente en el uso de la RAM en relación al servidor de aplicaciones JBoss

AS

3.4.1.3. Uso de RED para una concurrencia igual a 9

$\mu_1 = El\ uso\ de\ la\ \text{RED}\ del\ servidor\ de\ aplicaciones\ Glassfish$

$\mu_2 = El\ uso\ de\ la\ \text{RED}\ del\ servidor\ de\ aplicaciones\ JBoss\ AS$

Paso 1 – Tipo de prueba, establecer hipótesis y nivel de confianza

Hipótesis nula: El uso de la RED del servidor de aplicaciones GlassFish es igual al uso del RED del servidor de aplicaciones JBoss AS

$H_0 : \mu_1 = \mu_2$

Hipótesis alternativa: El uso del RED del servidor de aplicaciones GlassFish debe ser menor que el uso del RED del servidor de aplicaciones JBoss AS, para decir que es más eficiente en el consumo de la RED

$H_a : \mu_1 < \mu_2 \Rightarrow Prueba\ de\ cola\ izquierda$

Nivel de confianza

$\alpha = 0.03$

Paso 2 – Seleccionar distribución de probabilidad y valor crítico

Distribución normal por que $n > 30$ y se desconoce la desviación estándar de la población

El valor crítico es $z = -2.17$

Paso 3 – Calcular el error estándar y estandarizar las medias de la muestras

Como no se conoce la desviación estándar de la población se va ha decir

que la desviación estándar de la muestra es igual a la de la población.

$$\sigma = S$$

Calcular el error estándar

$$\sigma_{\bar{x}_1 - \bar{x}_2} = \sqrt{\frac{\sigma_1^2}{n} + \frac{\sigma_2^2}{n}} = 36{,}3407$$

Calcular z estandarizado para las medias de las muestras

$$z = \frac{(\bar{x}_1 - \bar{x}_2) - (\bar{\mu}_1 - \bar{\mu}_2) H_0}{\sigma_{\bar{x}_1 - \bar{x}_2}}$$

$$(\bar{\mu}_1 - \bar{\mu}_2) H_0 = 0$$

$$z = -8{,}4997$$

Paso 4 – Gráfica de la distribución

Se toma en referencia la Figura 39: Gráfica de la distribución de la normal

Paso 5 – Comprobar estadístico muestral con valores críticos

Se comprueba que $-8{,}4997 < -2.17$ y cae fuera de la zona donde se aceptación la hipótesis nula H_0

Por último se rechaza la hipótesis nula con un ERROR TIPO I con una probabilidad de cometer igual a α

Se puede concluir que: El servidor de aplicaciones GlassFish es más eficiente en el uso de la RED en relación al servidor de aplicaciones JBoss AS

3.4.1.4. Uso de CPU para una concurrencia igual a 200

$$\mu_1 = El\ uso\ de\ CPU\ del\ servidor\ de\ aplicaciones\ Glassfish$$

$\mu_2 = El\ uso\ de\ CPU\ \ del\ servidor\ de\ aplicaciones\ JBoss\ AS$

Paso 1 – Tipo de prueba, establecer hipótesis y nivel de confianza

Hipótesis nula: El uso del CPU del servidor de aplicaciones GlassFish es igual al uso del CPU del servidor de aplicaciones JBoss AS

$H_0 : \mu_1 = \mu_2$

Hipótesis alternativa: El uso del CPU del servidor de aplicaciones GlassFish debe ser menor que el uso del CPU del servidor de aplicaciones JBoss AS, para decir que es más eficiente en el consumo de la CPU

$H_a : \mu_1 < \mu_2 \Rightarrow Prueba\ de\ cola\ izquierda$

Nivel de confianza

$\alpha = 0.03$

Paso 2 – Seleccionar distribución de probabilidad y valor crítico

Distribución normal por que $n > 30$ y se desconoce la desviación estándar de la población

El valor crítico es $z = -2.17$

Paso 3 – Calcular error estándar y estandarizar las medias de la muestras

Como no se conoce la desviación estándar de la población se va ha decir que la desviación estándar de la muestra es igual a la de la población.

$\sigma = S$

Calcular el error estándar

$$\sigma_{\bar{x}_1-\bar{x}_2}=\sqrt{\frac{\sigma_1^2}{n}+\frac{\sigma_2^2}{n}}=0{,}4416$$

Calcular z estandarizado para las medias de las muestras

$$z=\frac{(\bar{x}_1-\bar{x}_2)-(\bar{\mu}_1-\bar{\mu}_2)H_0}{\sigma_{\bar{x}_1-\bar{x}_2}}$$

$$(\bar{\mu}_1-\bar{\mu}_2)H_0=0$$

$$z=-33{,}4971$$

Paso 4 – Gráfica de la distribución

Se toma en referencia la Figura 39: Gráfica de la distribución de la normal

Paso 5 – Comprobar estadístico muestral con valores críticos

Se comprueba que $-33{,}4971<-2.17$ y cae fuera de la zona donde se aceptación de la hipótesis nula H_0

Por último se rechaza la hipótesis nula con un ERROR TIPO I con una probabilidad de cometer igual a α

Se puede concluir que: El servidor de aplicaciones GlassFish es más eficiente en el uso de la CPU en relación al servidor de aplicaciones JBoss AS

3.4.1.5. Uso de RAM para una concurrencia igual a 200

$\mu_1 = El\ uso\ de\ la\ \text{RAM}\ del\ servidor\ de\ aplicaciones\ Glassfish$

$\mu_2 = El\ uso\ de\ la\ \text{RAM}\ del\ servidor\ de\ aplicaciones\ JBoss\ AS$

Paso 1 – Tipo de prueba, establecer hipótesis y nivel de confianza

Hipótesis nula: El uso de la RAM del servidor de aplicaciones GlassFish es igual al uso del RAM de del servidor de aplicaciones JBoss AS

$H_0 : \mu_1 = \mu_2$

Hipótesis alternativa: El uso del RAM del servidor de aplicaciones GlassFish debe ser menor que el uso de la RAM del servidor de aplicaciones JBoss AS, para decir que es más eficiente en el consumo de la RAM

$H_a : \mu_1 < \mu_2 \Rightarrow Prueba\ de\ cola\ izquierda$

Nivel de confianza

$\alpha = 0.03$

Paso 2 – Seleccionar distribución de probabilidad y valor crítico

Distribución normal por que $n > 30$ y se desconoce la desviación estándar de la población

El valor crítico es $z = -2.17$

Paso 3 – Calcular el error estándar y estandarizar las medias de la muestras

Como no se conoce la desviación estándar de la población se va ha decir que la desviación estándar de la muestra es igual a la de la población.

$$\sigma = S$$

Calcular el error estándar

$$\sigma_{\bar{x}_1 - \bar{x}_2} = \sqrt{\frac{\sigma_1^2}{n} + \frac{\sigma_2^2}{n}} = 0{,}2453$$

Calcular z estandarizado para las medias de las muestras

$$z = \frac{(\bar{x}_1 - \bar{x}_2) - (\bar{\mu}_1 - \bar{\mu}_2) H_0}{\sigma_{\bar{x}_1 - \bar{x}_2}}$$

$$(\bar{\mu}_1 - \bar{\mu}_2) H_0 = 0$$

$$z = 62{,}6336$$

Paso 4 – Gráfica de la distribución

Se toma en referencia la Figura 39: Gráfica de la distribución de la normal

Paso 5 – Comprobar estadístico muestral con valores críticos

Se comprueba que $62{,}6336 > -2.17$ y cae dentro de la zona donde se aceptación de la hipótesis la H_0

Por último no se debe rechazar la hipótesis nula porque no hay una diferencia significativa entre la media del uso de la RAM del servidor de GlassFish y la media del uso de la RAM del servidor JBoss.

Se puede concluir que: El servidor de aplicaciones GlassFish es de igual de eficiente en el consumo de la RAM en relación al servidor de aplicaciones JBoss AS

3.4.1.6. Uso de RED para una concurrencia igual a 200

$\mu_1 = El\ uso\ de\ la\ \text{RED}\ del\ servidor\ de\ aplicaciones\ Glassfish$

$\mu_2 = El\ uso\ de\ la\ \text{RED}\ del\ servidor\ de\ aplicaciones\ JBoss\ AS$

Paso 1 – Tipo de prueba, establecer hipótesis y nivel de confianza

Hipótesis nula: El uso de la RED del servidor de aplicaciones GlassFish es igual al uso de la RED de del servidor de aplicaciones JBoss AS

$H_0 : \mu_1 = \mu_2$

Hipótesis alternativa: El uso de la RED del servidor de aplicaciones GlassFish debe ser menor que el uso de la RED del servidor de aplicaciones JBoss AS, para decir que es más eficiente en el consumo de la RED

$H_a : \mu_1 < \mu_2 \Rightarrow Prueba\ de\ cola\ izquierda$

Nivel de confianza

$\alpha = 0.03$

Paso 2 – Seleccionar distribución de probabilidad y valor crítico

Distribución normal por que $n > 30$ y se desconoce la desviación estándar de la población

El valor crítico es $z = -2.17$

Paso 3 – Calcular el error estándar y estandarizar las medias de la muestras

Como no se conoce la desviación estándar de la población se va ha decir que la desviación estándar de la muestra es igual a la de la población.

$$\sigma = S$$

Calcular el error estándar

$$\sigma_{\bar{x}_1 - \bar{x}_2} = \sqrt{\frac{\sigma_1^2}{n} + \frac{\sigma_2^2}{n}} = 35{,}97$$

Calcular z estandarizado para las medias de las muestras

$$z = \frac{(\bar{x}_1 - \bar{x}_2) - (\bar{\mu}_1 - \bar{\mu}_2) H_0}{\sigma_{\bar{x}_1 - \bar{x}_2}}$$

$$(\bar{\mu}_1 - \bar{\mu}_2) H_0 = 0$$

$$z = -9{,}5075$$

Paso 4 – Gráfica de la distribución

Se toma en referencia la Figura 39: Gráfica de la distribución de la normal

Paso 5 – Comprobar estadístico muestral con valores críticos

Se comprueba que $-9{,}5075 < -2.17$ y cae fuera de la zona donde se aceptación de la hipótesis nula H_0

Por último se rechaza la hipótesis nula con un ERROR TIPO I con una probabilidad de cometer igual a α

Se puede concluir que: El servidor de aplicaciones GlassFish es más eficiente en el consumo de la RED en relación al servidor de aplicaciones JBoss AS

3.5. Interpretación de los resultados

Estadísticamente se llega al siguiente tabla de resumen:

$\mu_1 = El\ uso\ del\ recurso\ del\ servidor\ de\ aplicaciones\ Glassfish$

$\mu_2 = El\ uso\ de\ recurso\ del\ servidor\ de\ aplicaciones\ JBoss\ AS$

1. Con una concurrencia de 9 personas

Tabla IX: *Interpretación de los resultados con una concurrencia = 9*

	GlassFish	JBOSS AS	Demostración
CPU (50%)	50%	41.5463%	$H_a: \mu_1 < \mu_2$ Con ERROR TIPO I
RAM (40%)	40%	31.4332%	$H_a: \mu_1 < \mu_2$ Con ERROR TIPO I
RED (10%)	10%	6.8209%	$H_a: \mu_1 < \mu_2$ Con ERROR TIPO I
TOTAL (100%)	**100%**	**79.8004%**	

2. Con una concurrencia de 200 personas

Tabla X: *Interpretación de los resultados con una concurrencia = 200*

	GlassFish	JBoss AS	Demostración
CPU (50%)	50%	41.2828%	$H_a: \mu_1 < \mu_2$ Con ERROR TIPO I
RAM (40%)	40%	40%	$H_0: \mu_1 = \mu_2$ Con ERROR TIPO II
RED (10%)	10%	6.4813%	$H_a: \mu_1 < \mu_2$ Con ERROR TIPO I
TOTAL (100%)	**100%**	**87.7641%**	

De la Tabla IX se puede decir que el servidor GlassFish es superior que el servidor JBoss AS.

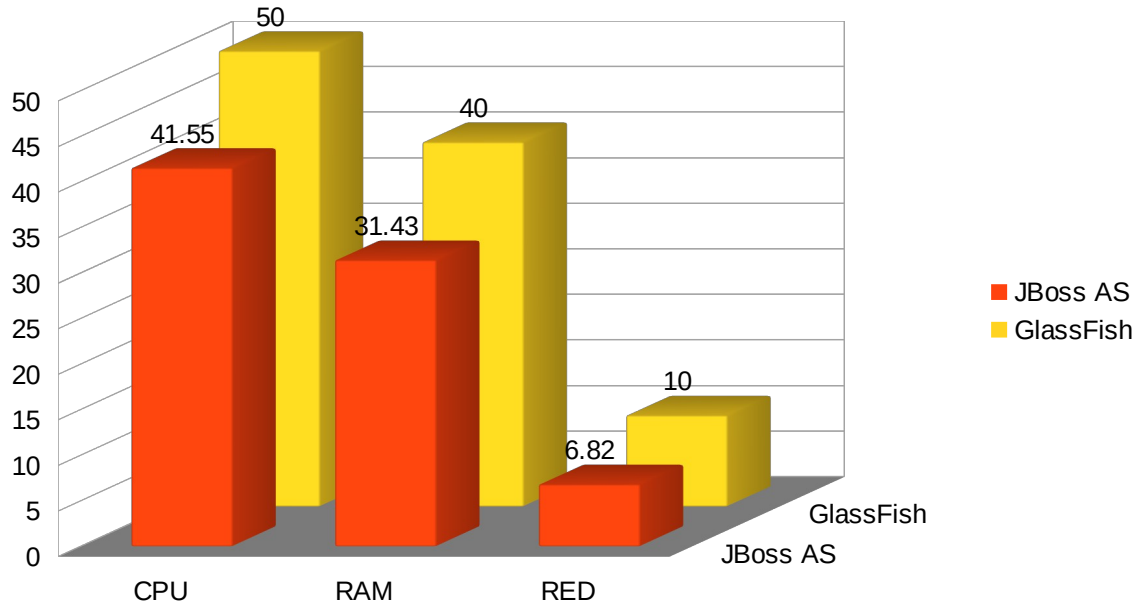


Figura 40: *Resultado final cuando la concurrencia igual a 9*

Realizado por: Diego P. Tamayo Barriga

Conclusión: Estadísticamente se ha llegado a la conclusión que para una concurrencia igual a 9, existe diferencia significativa entre la media de GlassFish y JBoss AS en el consumo de CPU, RAM y RED.

GlassFish cumple con todos los parámetros de esta prueba y se lleva el 100% en efectividad en el consumo de recursos, mientras que el servidor de aplicación JBoss AS tiene una efectividad del 79,8% de efectividad a relación de GlassFish.

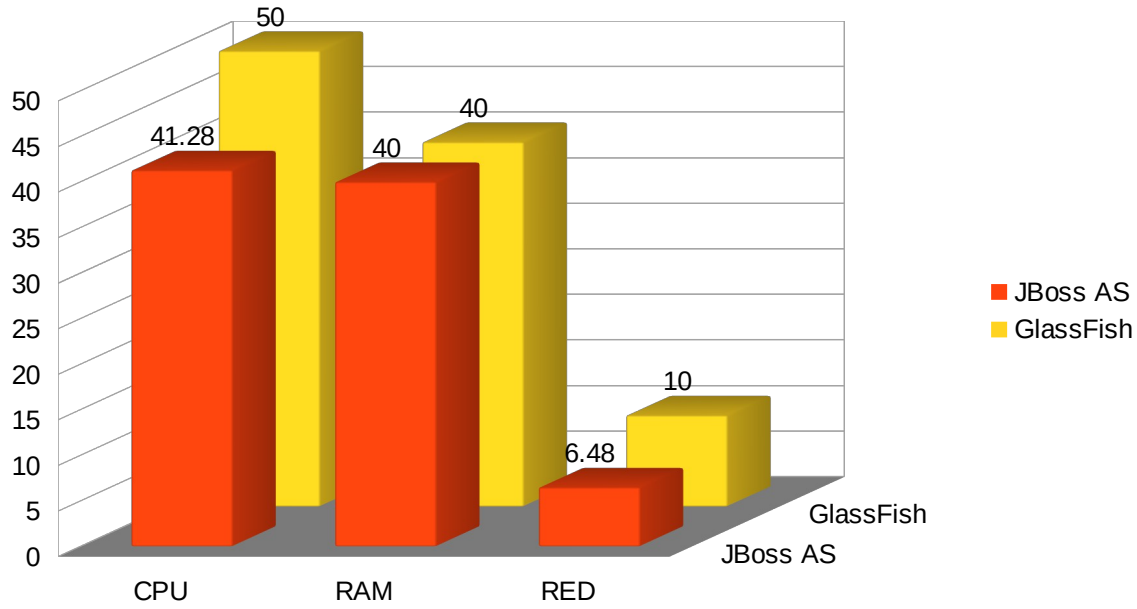


Figura 41: *Resultado final cuando la concurrencia igual a 200*

Realizado por: Diego P. Tamayo Barriga

Conclusión: Estadísticamente se ha llegado a la conclusión que para una concurrencia igual a 200, existe diferencia significativa entre la media de GlassFish y JBoss AS en el consumo de CPU y RED; mientras que en el consumo de RAM no existe diferencia signigicativa.

GlassFish cumple con todos los parámetros de esta prueba y se lleva el 100% en efectividad en el consumo de recursos, mientras que el servidor tiene una efectividad del 87,76% de efectividad a relación de GlassFish.

3.5.1. Comprobación de la hipótesis

Hipótesis

El servidor de aplicaciones GlassFish es más eficiente en el consumo de recursos en relación al servidor de aplicaciones JBoss.

Causa

El estudio estadístico de los servidores de aplicaciones.

Efecto

Seleccionar el más eficiente para el desarrollo del Módulo de Catálogos para el Sistema de Recursos Humanos.

Operación Conceptual

Tabla XI: *Operación Conceptual*

Variables	Tipo
Tipo de Servidor	Independiente
Uso del CPU	Dependiente
Uso de la RAM	Dependiente
Uso de la RED	Dependiente

Operacionalización Metodológica

Tabla XII: *Operacionalización metodológica*

Variable	Indicador	Técnicas	Fuente de Verificación
Tipo de Servidor	Funcionamiento. Envió de paquetes. Uso de recursos.	Observación	Pruebas Prototipo
Uso del CPU	Rendimiento	Experimentación Observación	Envió de Paquetes
Uso de la RAM	Rendimiento	Experimentación Observación	Envío de Paquetes
Uso de la RED	Rendimiento	Experimentación Observación	Envió de Paquetes

Conclusión Final

Utilizando la Estadística Descriptiva y Estadística Inferencial se puede concluir que el servidor de aplicaciones GlassFish es más eficiente en el consumo de recursos en relación al servidor JBoss AS, por lo que se ha

elegido para la implementación del Módulo de Catálogos del Sistema de Recursos Humanos.

3.6. Trabajos Futuros

Actualmente se encuentra una amplia variedad de plataformas y arquitecturas a elegir. A la hora de decantarse por una solución u otra, hay que tener en cuenta una serie de factores en función de cuales sean nuestras prioridades. Entre esos factores se puede encontrar la arquitectura del sistema operativo con el que se realiza las pruebas, el entorno de ejecución de la Plataforma Java, el sistema operativo libre con un sistema operativo propietario, a esto se le puede sumar la idea de virtualización para realizar una comparación entre equipos físicos y virtualizados.

3.6.1. Otras factores a estudiar

Los factores que no comprende el presente estudio y podría ser objeto de estudio en futuros u otros estudios son:

- Comparación de rendimiento entre plataformas Java Oracle y OpenJDK.
- Comparación de rendimiento entre arquitecturas de 32 y 64 bits.
- Comparativa de rendimiento de servidores de aplicación libres y servidores de aplicación propietarios para la Plataforma de Java, Enterprise Edition.
- Comparativa de rendimiento de un servidor ejecutándose en un sistema operativo Linux y un sistema operativo propietario Microsoft

Windows

- Comparativa de rendimiento en un ambiente emulado con Hiper-V, VMware, Xen o Oracle VM VirtualBox y un ambiente físico.

3.6.2. Otras plataformas a estudiar

También sería muy interesante realiza el estudio de otras plataformas que podrían ajustarse muy bien a los requerimientos:

- Comparativa entre servidores para la plataforma Java, Enterprise Edition y el servidor de Microsoft Internet Information Server.
- Comparativa entre servidores para la plataforma Java, Enterprise Edition y el servidor de Apache HTTP para PHP.
- Comparativa entre el servidor de Apache HTTP y el servidor Microsoft Internet Information Server.

Capítulo IV

4. Implementar el módulo de catálogos del sistema de descripción y validación de puestos de trabajo para el Departamento de Recursos Humanos

El Departamento de Sistemas y Telemática de la Escuela Superior Politécnica de Chimborazo, siendo está un departamento que cubre las necesidades en cuánto al desarrollo de sistemas para las diferentes unidades de la universidad. Está siendo una entidad pública tiene la obligación de la utilización del Software Libre para cubrir todos los requisitos para su desarrollo.

El Departamento de Recursos Humanos tiene la necesidad de un sistema que permita la descripción, clasificación y validación de un puesto de trabajo. Este permitirá un mejor control entre las distintas unidades como el Rectorado, Recursos Financieros y Recursos Humanos, mediante el sistema los procesos administrativos se realizarán de una forma ordenada y lógica.

4.1. Solución planteada

Para la creación de la aplicación web ha surgido un numeroso conjunto de herramientas y lenguajes de programación; entre los que destacan la plataforma Java que se encuentra entre las primeras posiciones según la empresa TIOBE Software y además es una herramienta de desarrollo distribuida bajo la filosofía de Software Libre.

La plataforma cuenta con un conjunto de herramientas que facilita el desarrollo, cuenta con un conjunto de frameworks que ayuda para la solución así como una gran comunidad a nivel mundial para la ayuda en temas que se compliquen al momento del desarrollo.

Así también se utilizará patrones de desarrollo que aportar calidad y mejor mantenimiento al software creado.

4.1.1. Pre-análisis de la solución

Cómo ya se ha mencionado se utilizará la plataforma Java, específicamente la plataforma Java, Enterprise Edition que está diseñada para cubrir todas las necesidades a lo que se refiere al desarrollo de aplicaciones web ya que cuenta con una especificación para la creación de dichas aplicaciones.

Entre las herramientas para la creación se utilizará el entorno de desarrollo Netbeans, el cual cuenta con una gran cantidad de asistentes para la ayuda al desarrollador y mantiene una interfaz muy intuitiva para el desarrollo de un numeroso conjunto de aplicaciones desde aplicativos móviles hasta un aplicativo web complejo.

Dentro de los patrones utilizados se usa MVC para la creación de la aplicación web, este patrón enseña la forma como debe estar estructurada la aplicación para su mejor comprensión.

4.1.2. Solución propuesta

La creación del sistema web se lo realizará con JavaServer Faces que maneja el patrón de diseño MVC desde su creación, para ayudar con la parte gráfica y así crear una interfaz amigable al usuario; este se apoyara con PrimeFaces el cuál es un framework de componentes que facilita el desarrollo.

Entre los patrones que se utiliza en el desarrollo está:

- Composite
- Facade
- Decorator

4.1.2.1. Composite

Sirve para construir objetos complejos a partir de otros más simples y similares entre sí, gracias a la composición recursiva y a una estructura de árbol.

Esto simplifica el tratamiento de los objetos creados, ya que al poseer todos ellos una interfaz común, se tratan todos de la misma manera.

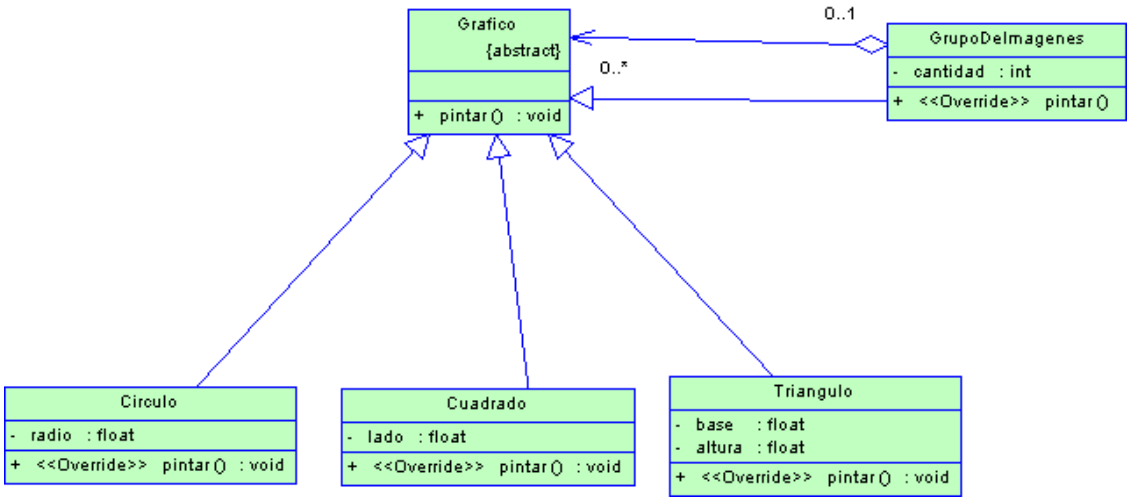


Figura 42: *Patrón de diseño Composite*

Fuente: http://upload.wikimedia.org/wikipedia/commons/b/be/Uml_composite.png

4.1.2.2. Facade

Este patrón se ocupa cuando se necesita proporcionar una interfaz simple para un subconjunto complejo o cuando se quiere estructurar varios subsistemas en capas, ya que las fachadas serian el punto de entrada a cada nivel.

Otro escenario es cuando surge la necesidad de desacoplar un sistema de sus clientes y de otros subsistemas, haciendo más independiente, portable y reutilizable.

Entre las principales ventajas del patrón es que para modificar la clase de los subsistemas, sólo hay que realizar cambios en la interfaz/fachada y los clientes pueden permanecer ajenos a ello. Además y cómo se mencionó anteriormente los clientes no necesitan conocer las clases que hay tras dicha interfaz.

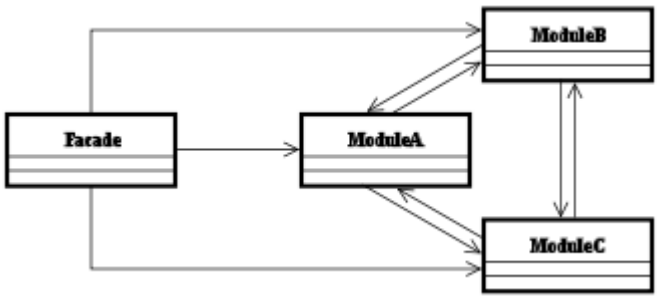


Figura 43: *Patrón de diseño Facade*

Fuente:
http://upload.wikimedia.org/wikipedia/commons/thumb/d/d4/Facade_UML_class_diagram.svg/350px-Facade_UML_class_diagram.svg.png

4.1.2.3. Decorator

Este patrón responde a la necesidad de añadir dinámicamente funcionalidad a un objeto.

Esto permite no tener que crear sucesivas clases que hereden de la primera incorporación la nueva funcionalidad, sino otras que la implementan y se asocian a la primera.

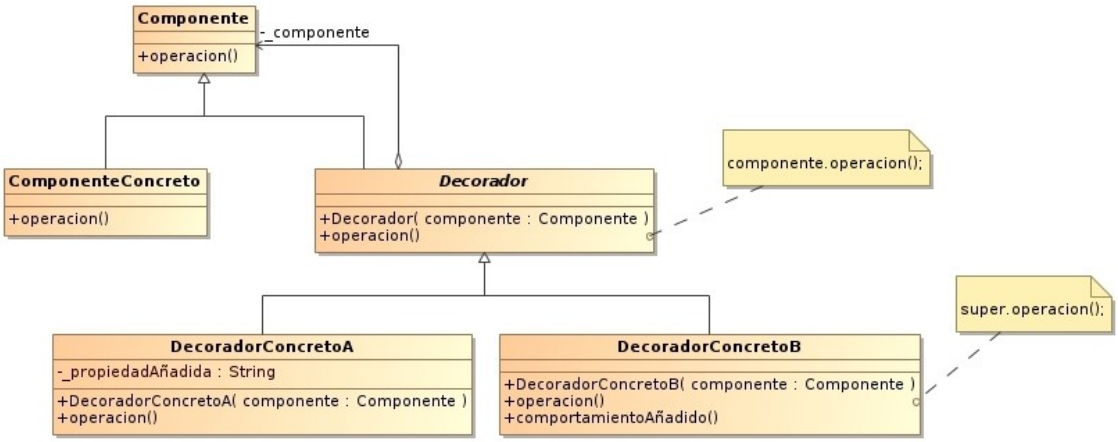


Figura 44: *Patrón de diseño Decorator*

Fuente: http://upload.wikimedia.org/wikipedia/commons/b/be/DecoradorConcretoF.jpg

4.1.3. Pre-requisitos para la solución

Para el desarrollo se cuenta con la configuración está se detalla en el capítulo 3.1 Configuración del sistema:

- 3.1.1 Configuración Hardware tanto para el desarrollo y su posterior uso en las pruebas de benchmark para dar a conocer cuál de los servidores es el más eficiente al momento de su implementación.
- 3.1.2 Configuración Software

4.2. Metodología del Desarrollo del Módulo

Para su implementación se uso la Metodología SCRUM (Ver el capítulo 2.8 Descripción de la Metodología SCRUM), el cuál es un modelo de desarrollo adaptable, orientado a las personas y se basa en la construcción de incrementos funcionales del producto final.

4.2.1. Producto Backlog

En este espacio se detallará los requerimientos en global para el Módulo de Catálogos del Sistema de Recursos Humanos de la Escuela Superior Politécnica de Chimborazo, estos constaran de un identificador, descripción y prioridad. No es importante que los requerimientos estén ordenados por algún criterio.

Tabla XIII: *Product Backlog*

Nº	Descripción	Prioridad
1	Maquetación de Pantalla Principales y Secundarias	1
2	Creación de las hojas de estilo para las distintas pantallas	4
3	Creación de las plantillas JavaServer Faces para las distintas pantallas	3
4	Creación del diagrama relacional para la base de datos	1
5	Creación de la base de datos	2
6	Creación de la capa de persistencia (Modelo)	2
7	Definir entidades que corresponden al módulo de catálogos	3
8	Creación de los controladores de entidades del módulo de catálogos (Controlador)	3
9	Creación de las vistas de las entidades del módulo de catálogos	3
10	Creación del sistema de escritura de URL	4

4.2.2. Requisitos no Funcionales

Un requisito no funcional o atributo de calidad es un requisito que especifica criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos, ya que éstos corresponden a los requisitos funcionales.

Amigabilidad

Cualidad de una interfaz de programa que por su forma de interactuar con el usuario es considerada de fácil uso.

Fiabilidad

La fiabilidad se define como la probabilidad de que un bien funcione adecuadamente durante un período determinado bajo condiciones operativas específicas

Usabilidad

Se refiere a la capacidad de un software de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso.

Mantenibilidad

La facilidad con la que un sistema o componente software puede ser modificado para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarse a cambios en el entorno.

Portabilidad

Se define como la característica que posee un software para ejecutarse en diferentes plataformas, el código fuente del software es capaz de reutilizarse en vez de crearse un nuevo código cuando el software pasa de una plataforma a otra. A mayor portabilidad menor es la dependencia del software con respecto a la plataforma.

Interoperabilidad

Es la habilidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada

Escalabilidad

Es la propiedad deseable de un sistema, una red o un proceso, que indica su habilidad para reaccionar y adaptarse sin perder calidad, o bien manejar el crecimiento continuo de trabajo de manera fluida, o bien para estar preparado para hacerse más grande sin perder calidad en los servicios ofrecidos.

Concurrencia

Es la propiedad de los sistemas que permiten que múltiples procesos sean ejecutados al mismo tiempo y que potencialmente puedan interactuar entre sí.

Los procesos concurrentes pueden ser ejecutados realmente de forma simultánea.

4.2.3. Pila del Sprints

4.2.3.1. Sprint 0 o Análisis previo

En este sprint tuvo aproximadamente la duración de unos 15 días, en los que se realizó el levantamiento de los requerimientos.

Tabla XIV: *Sprint 0 o análisis previo*

Nº	Tarea	Tipo
1	Maquetación de Pantalla Principales y Secundarias	Análisis/Diseño Anexo C
2	Creación del diagrama relacional para la base de datos	Análisis/Diseño Anexo D

4.2.3.2. Sprint 1

En este sprint tuvo aproximadamente la duración de unos 15 días, este se centró en integrar la parte de la base de datos para la posterior utilización en las diferentes tareas.

Tabla XV: *Sprint 1*

Nº	Tarea	Tipo
1	Creación de la base de datos	Desarrollo
2	Creación de la capa de persistencia (Modelo)	Desarrollo Anexo E
3	Creación de las plantillas JavaServer Faces para las distintas pantallas	Desarrollo Anexo H

4	Definir entidades que corresponden al módulo de catálogos	Desarrollo Anexo E

4.2.3.3. Sprint 2

En este sprint tuvo aproximadamente la duración de unos 15 días, está se centra en el desarrollo del módulo de catálogos del Departamento de Recursos Humanos.

Tabla XVI: *Sprint 2*

Nº	**Tarea**	**Tipo**
1	Creación de los controladores de entidades del módulo de catálogos (Controlador)	Desarrollo Anexo G
2	Creación de las vistas de las entidades del módulo de catálogos	Desarrollo Anexo H
3	Creación del sistema de escritura de url	Desarrollo

CONCLUSIONES

- La utilización de servidores de aplicaciones libre para la creación de aplicaciones web robustas y distribuidas, son comparables con herramientas de software propietario llegando a obtener los mismos resultados requeridos en el desarrollo e implementación.
- El uso de patrones de diseño evita la búsqueda de soluciones a problemas conocidos y solucionados con anterioridad, así también permite la reutilización del código realizando el menor esfuerzo en el desarrollo y construcción de una aplicación web.
- En el análisis comparativo realizado, los parámetros seleccionados son parte de las características necesarias que los servidores para aplicaciones web deben tener; esto permitió, determinar cuál es el mejor servidor de aplicaciones para la plataforma Java EE.
- Aplicando la distribución normal con las muestras obtenidas en las pruebas de rendimiento que se realizaron, se pudo comprobar que el servidor de aplicaciones GlassFish es más eficiente en el consumo de los recursos en relación al servidor de aplicaciones JBoss AS.
- En el estudio estadístico que se realizó entre los tipos de servidores para la plataforma de Java EE, que son GlassFish y JBoss AS; existen diferencias significativas en el uso del CPU, RAM y RED, los resultados establece que para una concurrencia de 9 hilos simultáneos, el servidor GlassFish es 20.2% más eficiente en el consumo de los recursos, frente al servidor JBoss AS. Para una concurrencia de 200 hilos simultáneos, el servidor GlassFish sigue siendo más efectivo con un 12.24% en el uso de los recursos en

relación al servidor JBoss AS.

- El desarrollo del módulo de Catálogos del Sistema de Recursos Humanos se creó mediante la utilización del lenguaje de programación Java, un entorno de desarrollo integrado como Netbeans compatible con la plataforma Java EE, el servidor de base de datos PostgreSql y en un sistema operativo Linux; representando una reducción en costo, libertad de uso y distribución, y uso de formatos estándares; los cuales fomentan una libre competencia.

RECOMENDACIONES

- Se sugiere el empleo de frameworks en el desarrollo de aplicaciones web, ya que ayudan al desarrollo reduciendo tiempos y líneas de código.
- Se recomienda el uso de herramientas CASE, ya que aumenta la productividad en el desarrollo de software, reduciendo el costo de las mismas en término de tiempo.
- Se aconseja llevar a cabo todos los procedimientos que se detallan en este informe para el desarrollo de aplicaciones web utilizando la plataforma Java EE. Así también; se sugiere la utilización de las tecnologías que incorpora la plataforma las cuales son EJB (Enterprise JavaBeans), JPA (Java Persistence API), CDI (Context and Dependency Injection) y JSF (Java ServerFaces) para obtener un mejor desempeño de desarrollo, manejo de estándares, limpieza de código y el fácil mantenimiento del sistema.
- Se recomienda hacer uso del patrón de diseño MVC (Model – View - Controller) proporcionado por la tecnología JSF para el desarrollo de aplicaciones web, ya que permite la reutilización de código, separando las capas de desarrollo y es fácil de aprender.
- Se ha utilizado en el proyecto la metodología de desarrollo ágil SCRUM, obteniendo buenos resultados; por lo que se recomienda su uso.
- Se aconseja mantener una revisión del módulo de catálogos para

entender y comprender el uso de patrones de diseño en base a buenas prácticas de desarrollo, y la utilización de estándares de la plataforma Java EE.

- Se recomienda para la implementación del módulo de Catálogos del sistema de Recursos Humanos, el uso del servidor de aplicaciones GlassFish con una concurrencia de 9 o 200 hilos simultáneos ya que este es más eficientes en el uso de los componentes hardware.
- Para futuros trabajos de investigación se recomienda un análisis de comparación del servidor de aplicaciones GlassFish en ambientes virtualizados entre las distintas herramientas que se encuentran disponibles en el mercado o realizar un estudio similar al presente comparando los servidores de aplicaciones libres frente a servidores de aplicaciones propietarios para la plataforma Java EE.

RESUMEN

En esta investigación se ha hecho un análisis comparativo de los servidores GlassFish y JBoss AS para la plataforma Java Enterprise Edition aplicado al módulo de catálogos del Sistema de Recursos Humanos de la Escuela Superior Politécnica de Chimborazo.

Para la elaboración del análisis comparativo de los servidores se utilizó el método científico, el cuál plantea un problema, el servidor GlassFish es más eficiente en consumo de los recurso hardware en relación al servidor JBoss AS, para recopila la información se utiliza una herramienta de benchmarking que proporciona en tiempo real el consumo del cpu, consumo de memoria ram y el consumo de la red, y para el análisis la estadística descriptiva e inferencial. Se elaboró un ambiente parecido al que a posterior se va a implementar con un servidor Centos y la plataforma Java instalada para la ejecución de los servidores.

En el análisis comparativo se observó que el servidor GlassFish para una concurrencia igual a 9 es más eficiente en el consumo de recursos en relación a un 79,8% del servidor JBoss AS. Para una concurrencia igual a 200 el servidor GlassFish sigue siendo eficiente en el consumo de recursos en relación a un 87,76% del servidor JBoss AS.

En conclusión aplicando la distribución muestral de diferencia entre medias muestrales y la distribución normal con los medias obtenidas de las prueba de benchmarking se pudo comprobar estadísticamente el servidor más eficientes para la implementación del Módulo de Catálogos del Sistema de Recursos Humanos.

Se recomienda en la definición de estos parámetros de comparación que se realicé un análisis previo para determinar el servidor más eficiente tomando en cuenta de su uso que se dará a posterior.

ABSTRACT

This proposal has been developed through a comparative analysis from the servers Glassfish and JBoss for the Java Plataform Enterprise Edition. It has been applied to the catalogues of the human resources system located at Polytechnic of Chimborazo.

For elaboration the comparative analysis of both services the scientific method was used, It raises a problem; Glassfish server is more efficient in the consumption of hardware than JBoss AS. For collecting data benchmarking tool is used which provides in real time the consumption from cpu, ram disk and the network and also for the inferential and descriptive statistical analysis. It was elaborated in a similar enviroment that is going to be implemented with a Centos server and, the installation of Java Plataforma to execute the servers.

It was observed in the comparative analysis that the GlassFish server for a concurrency of 9. It is more efficient the consumtion of the resources in 79.8% of JBoss AS server. For a concurrency equals to 200 GlassFish server is still efficient in the consumption of the resources with 87,76% from JBoss AS server.

In conclusion when appluing the sample the different between sample and the normal distribution with the measurements obtained from the benchmarking test, it coul be checked the statistically with the more efficient server for implementing catalogues of the human resources system.

It is recommended to do a previous analysis of the definition of the comparison parameters for determining the most efficient server for using it afterwards.

GLOSARIO

Bytecode

Es un archivo binario que contiene un programa ejecutable que se produce por la compilación del código fuente escrito en la Plataforma Java.

CPU

La Unidad Central de Procesamiento o CPU (por el acrónimo en inglés de Central Processing Unit), o simplemente el procesador o microprocesador, es el componente del computador y otros dispositivos programables, que interpreta las instrucciones contenidas en los programas y procesa los datos.

Desviación Estándar

La desviación estándar o desviación típica es una medida de centralización o dispersión para variables de razón (ratio o cociente) y de intervalo, de gran utilidad en la estadística descriptiva.

Se define como la raíz cuadrada de la varianza. Junto con este valor, la desviación típica es una medida (cuadrática) que informa de la media de distancias que tienen los datos respecto de su media aritmética, expresada en las mismas unidades que la variable.

Error Tipo I

Se comete cuando la hipótesis nula es verdadera y como consecuencia se rechaza.

Error Tipo II

Se comete cuando la hipótesis nula es falsa y como consecuencia del contraste se acepta.

Journaling

Es un registro diario en el que se almacena la información necesaria para restablecer los datos afectados por la transacción en caso que falle.

Media

En matemáticas y estadística, la media aritmética (también llamada promedio o simplemente media) de un conjunto finito de números es igual a la suma de todos sus valores dividida entre el número de sumandos. Cuando el conjunto es una muestra aleatoria recibe el nombre de media muestral siendo uno de los principales estadísticos muestrales.

Expresada de forma más intuitiva, podemos decir que la media (aritmética) es la cantidad total de la variable distribuida a partes iguales entre cada observación.

Middleware

Es un software que asiste a una aplicación para interactuar o comunicarse

con otras aplicaciones, software, redes, hardware o sistemas operativos.

Mosaic

Fue el primer navegador gráfico disponible. Desarrollado por NCSA

Oak

Era un roble que había fuera de la oficina de Gosling, el creador del lenguaje de programación Java.

Proxy

Es un programa o dispositivos que realiza una acción en representación de otro.

RAM

La memoria principal o RAM (Random Access Memory, Memoria de Acceso Aleatorio) es donde el computador guarda los datos que está utilizando en el momento presente. El almacenamiento es considerado temporal por que los datos y programas permanecen en ella mientras que la computadora este encendida o no sea reiniciada.

Smalltalk

Lenguaje de Programación Orientada a Objetos en la que se basó Java para su construcción.

Varianza

En teoría de probabilidad, la varianza de una variable aleatoria es una medida de su dispersión definida como la esperanza del cuadrado de la desviación de dicha variable respecto a su media.

Está medida en unidades distintas de las de la variable. Por ejemplo, si la variable mide una distancia en metros, la varianza se expresa en metros al cuadrado. La desviación estándar, la raíz cuadrada de la varianza, es una medida de dispersión alternativa expresada en las mismas unidades.

Capitulo V

BIBLIOGRAFIA

1. **BIOCCA, S., Y OTROS.,** A Short Course on the Basic., 5ª, ed., Michigan - United States., Addison-Wesley., 2013., Pag. 31-280, 371-443.

2. **BRUCE, E.,** Thinking in Java., 4ª, ed., Massachusetts – United States., Prentice Hall., 2006., Pag. 15-91, 145-564, 725-795.

3. **DEITEL, P., Y OTROS.,** Java How to program., 9ª, ed., Massachusetts - United States., Prentice Hall., 2013., Pag. 71-463, 672-708, 829-930.

4. **FREEMAN, E., Y OTROS.,** Head First Design Patterns., 1ª, ed., United States., O'Really Media, Inc., 2004., Pag. 110, 281.

5. **GOSLING, J., Y OTROS.,** The Javath Language Specification., 3ª, ed., California – United States., Addison-Wesley., 2005., Pag. 33-73, 153-307.

6. **JENDROCK, E., Y OTROS.,** The Java EE 6 Tutorial., 2ª, ed., California - United States., 2013., Pag. 103-334, 433-687.

7. **PALACIOS, J.,** Scrum Manager: Gestión de Proyectos., 2ª, ed., Creative., 2010., Pag. 44-45, 49-86.

8. **SOLFA, F.,** Benchmarking en el sector público., La Plata – Argentina., 2012., Pag. 9, 12-15.

BIBLIOGRAFIA INTERNET

1. **ab – Apache HTTP server benchmarking tool**

   http://httpd.apache.org/docs/2.2/programs/ab.html
   2013-06-09

2. **INTERFAZ DE PERSISTENCIA JAVA (JPA) – ENTIDADES Y MANAGERS**

   http://www.lab.inf.uc3m.es/~a0080802/RAI/jpa.html

   2013-09-20

3. **INTRODUCCION A LOS SERVIDORES DE APLICACIONES**

http://www.jtech.ua.es/j2ee/2003-2004/abierto-j2ee-2003-2004/sa/sesion1-apuntes.htm

2013-04-19

4. **JAVA EE: SEGURIDAD EN APLICACIONES WEB**

http://jdiezfoto.es/informatica/java-ee-seguridad-en-aplicaciones-web-i/

2013-08-03

5. **JAVA SE APPLICATION DESIGN WITH MVC**

http://www.oracle.com/technetwork/articles/javase/index-142890.html

2013-06-09

6. **JSF – JAVA SERVER FACES**

http://osl2.uca.es/wikiii/index.php/JSF#Introducci.C3.B3n
2013-07-11

7. **JSF 2.0: MANAGED BEANS**

http://anadreamy.wordpress.com/2011/10/21/jsf-2-0-managed-beans/

2013-10-21

8. **MIDIENDO EL RENDIMIENTO DEL SISTEMA CON SAR**

http://mundogeek.net/traducciones/midiendo-el-rendimiento-del-sistema-con-SAR.htm

2013-06-03

9. **PERFIL COMPLETO JAVA PLATAFORM, ENTERPRISE EDITION**

http://publib.boulder.ibm.com/wasce/V3.0.0/es/java-ee-6-certified.html

2013-07-01

10. **SUPERVICION DE ACTIVIDADES DEL SISTEMA (SAR)**

http://docs.oracle.com/cd/E24842_01/html/E23086/spmonitor-8.html

2013-08-01

11. **DATOS MUESTRALES**

CDs., TAMAYO D., ESPOCH.,

2013-10-22

ANEXOS

A. Desarrollo de Cálculos Probabilísticos

En este anexo se muestra las fórmulas aplicadas y los valores que se establecieron para tomar la decisiones.

Los valores que se van a encontrar son la media, la varianza, desviación estándar e intervalo de confianza para cada una de las pruebas con una concurrencia igual a 9 y 200.

De los datos anteriores enunciados en capítulo 3.3.2 Métodos, técnicas y procedimientos se tiene que:

$$nivel\ de\ confianza = 0.97\,\%$$

$$e = margen\ de\ error = 0.03\,\%$$

$$z = para\ el\ nivel\ de\ confianza\ del\ 0.97\ porciento\ ,\ z\ es = 2.17$$

Cuando no se conoce la desviación estándar de la población se usa el valor constante de la desviación estándar.

$$\sigma = desviación\ estándar = 0.5$$

Cálculo de la muestra para cuando no se conoce el tamaño de la población

$$n = \frac{(z^2 * \sigma^2)}{e^2}$$

$$n = 1309$$

Fórmula de la Media

$$\bar{x} = \sum_{i=1}^{n} \frac{x}{n}$$

Fórmula de la Varianza

$$S^2 = \sum_{i=1}^{n} \frac{x_i - \bar{x}}{n}$$

Fórmula de la Desviación Estándar

$$S_{\bar{x}} = \sqrt{S^2}$$

Concurrencia igual a 9 para CPU

Resumen para el servidor GlassFish

Cálculos: Estadística Descriptiva	
Valor de Z (97% Nivel de Confianza) :	2.1700
Muestra:	1309.0000
Media:	70.4930
Varianza:	166.3548
Desviación Estándar:	12.8979

Realizado por: Diego P. Tamayo Barriga

Resumen para el servidor JBoss AS

Valor de Z (97% Nivel de Confianza) :	2.1700
Muestra:	1309.0000
Media:	84.8371
Varianza:	80.0901
Desviación Estándar:	8.9493

Realizado por: Diego P. Tamayo Barriga

Concurrencia igual a 9 para RAM

Resumen para el servidor GlassFish

Cálculos: Estadística Descriptiva	
Valor de Z (97% Nivel de Confianza) :	2.1700
Muestra:	1309.0000
Media:	54.3815
Varianza:	78.6966
Desviación Estándar:	8.8711

Realizado por: Diego P. Tamayo Barriga

Resumen para el servidor JBoss AS

Cálculos: Estadística Descriptiva	
Valor de Z (97% Nivel de Confianza) :	2.1700
Muestra:	1309.0000
Media:	69.2026
Varianza:	1.4115
Desviación Estándar:	1.1881

Realizado por: Diego P. Tamayo Barriga

Concurrencia igual a 9 para RED

Resumen para el servidor GlassFish

Cálculos: Estadística Descriptiva	
Valor de Z (97% Nivel de Confianza) :	2,1700
Muestra:	1309,0000
Media:	662,7350
Varianza:	1032264,4687
Desviación Estándar:	1016,0042

Realizado por: Diego P. Tamayo Barriga

Resumen para el servidor JBoss AS

Cálculos: Estadística Descriptiva	
Valor de Z (97% Nivel de Confianza) :	2,1700
Muestra:	1309,0000
Media:	971,6179
Varianza:	696457,3481
Desviación Estándar:	834,5402

Realizado por: Diego P. Tamayo Barriga

Concurrencia igual a 200 para CPU

Resumen para el servidor GlassFish

Cálculos: Estadística Descriptiva	
Valor de Z (97% Nivel de Confianza) :	2.1700
Muestra:	1309.0000
Media:	70.0464
Varianza:	175.1225
Desviación Estándar:	13.2334

Realizado por: Diego P. Tamayo Barriga

Resumen para el servidor JBoss AS

Cálculos: Estadística Descriptiva	
Valor de Z (97% Nivel de Confianza) :	2,1700
Muestra:	1309,0000
Media:	84,8371
Varianza:	80,0901
Desviación Estándar:	8,9493

Realizado por: Diego P. Tamayo Barriga

Concurrencia igual a 200 para RAM

Resumen para el servidor GlassFish

Cálculos: Estadística Descriptiva	
Valor de Z (97% Nivel de Confianza) :	2,1700
Muestra:	1309,0000
Media:	69,7444
Varianza:	0,0572
Desviación Estándar:	0,2392

Realizado por: Diego P. Tamayo Barriga

Resumen para el servidor JBoss AS

Cálculos: Estadística Descriptiva	
Valor de Z (97% Nivel de Confianza) :	2,1700
Muestra:	1309,0000
Media:	54,3815
Varianza:	78,6966
Desviación Estándar:	8,8711

Realizado por: Diego P. Tamayo Barriga

Concurrencia igual a 200 para RED

Resumen para el servidor GlassFish

Cálculos: Estadística Descriptiva	
Valor de Z (97% Nivel de Confianza) :	2,1700
Muestra:	1309,0000
Media:	629,6344
Varianza:	997178,6391
Desviación Estándar:	998,3157

Realizado por: Diego P. Tamayo Barriga

Resumen para el servidor JBoss AS

Cálculos: Estadística Descriptiva	
Valor de Z (97% Nivel de Confianza) :	2,1700
Muestra:	1309,0000
Media:	971,6179
Varianza:	696457,3481
Desviación Estándar:	834,5402

Realizado por: Diego P. Tamayo Barriga

B. Prueba de 5 pasos para la demostración de hipótesis

Paso 1

a) Prueba de 2 colas o 1 cola

b) Establecer la hipótesis

c) Seleccionar un nivel de significancia

Paso 2

a) Escoger la distribución estadística (Normal o t Student)

b) Encontrar valores críticos

Paso 3

a) Calcular el Error Estándar

b) Convertir un valor observado en un valor estandarizado

Paso 4

a) Hacer bosquejo de la distribución

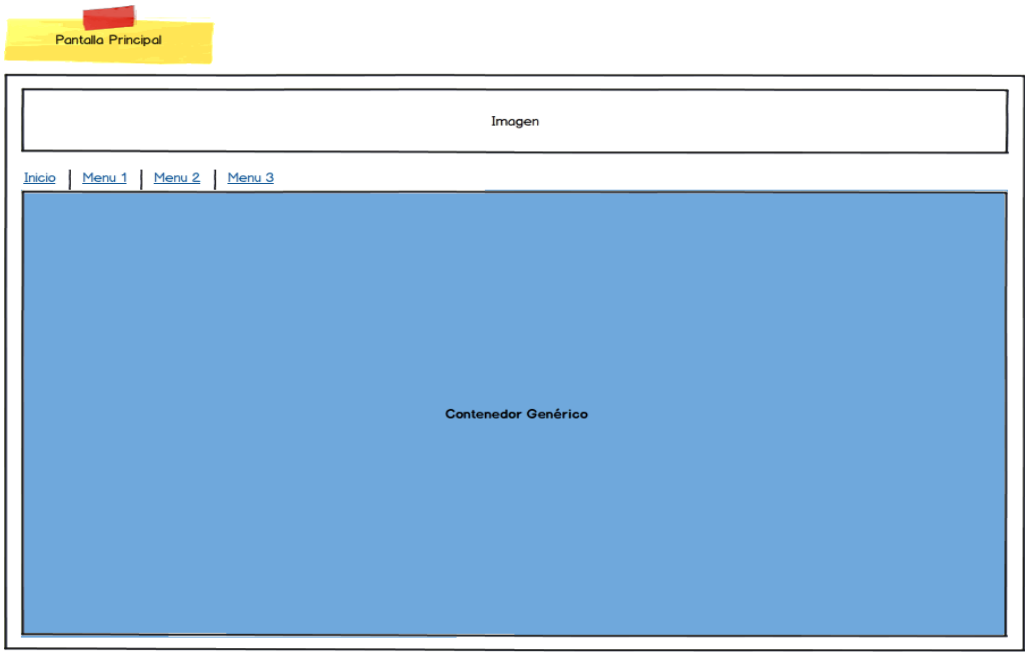
b) Marcar posición del valor estandarizado

Paso 5

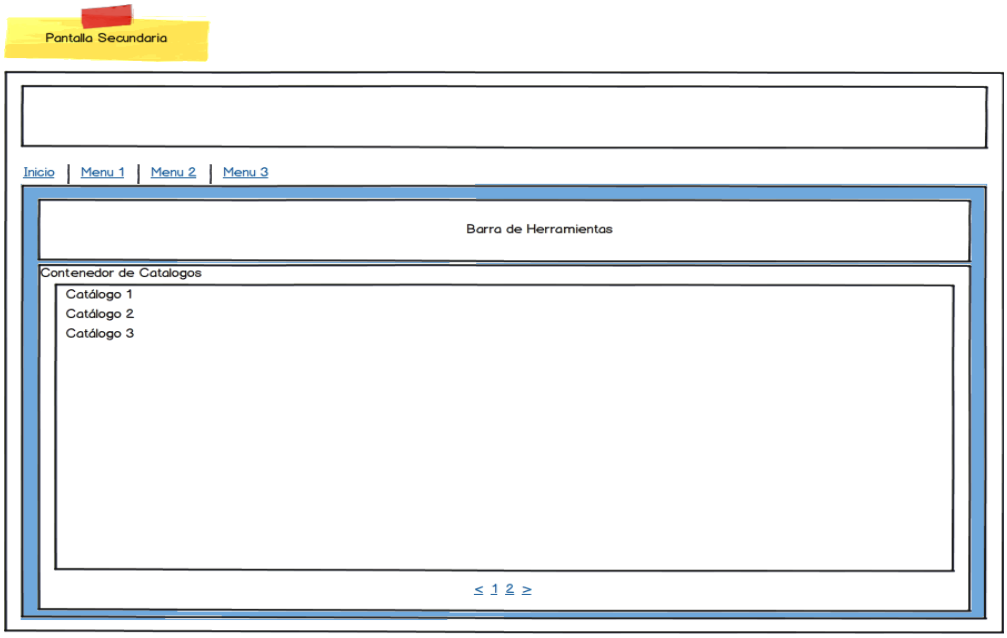
a) Compara estadístico muestral con valores críticos.

b) Llegar a la conclusión final

C. Maquetación de Pantallas Principales y Secundarias

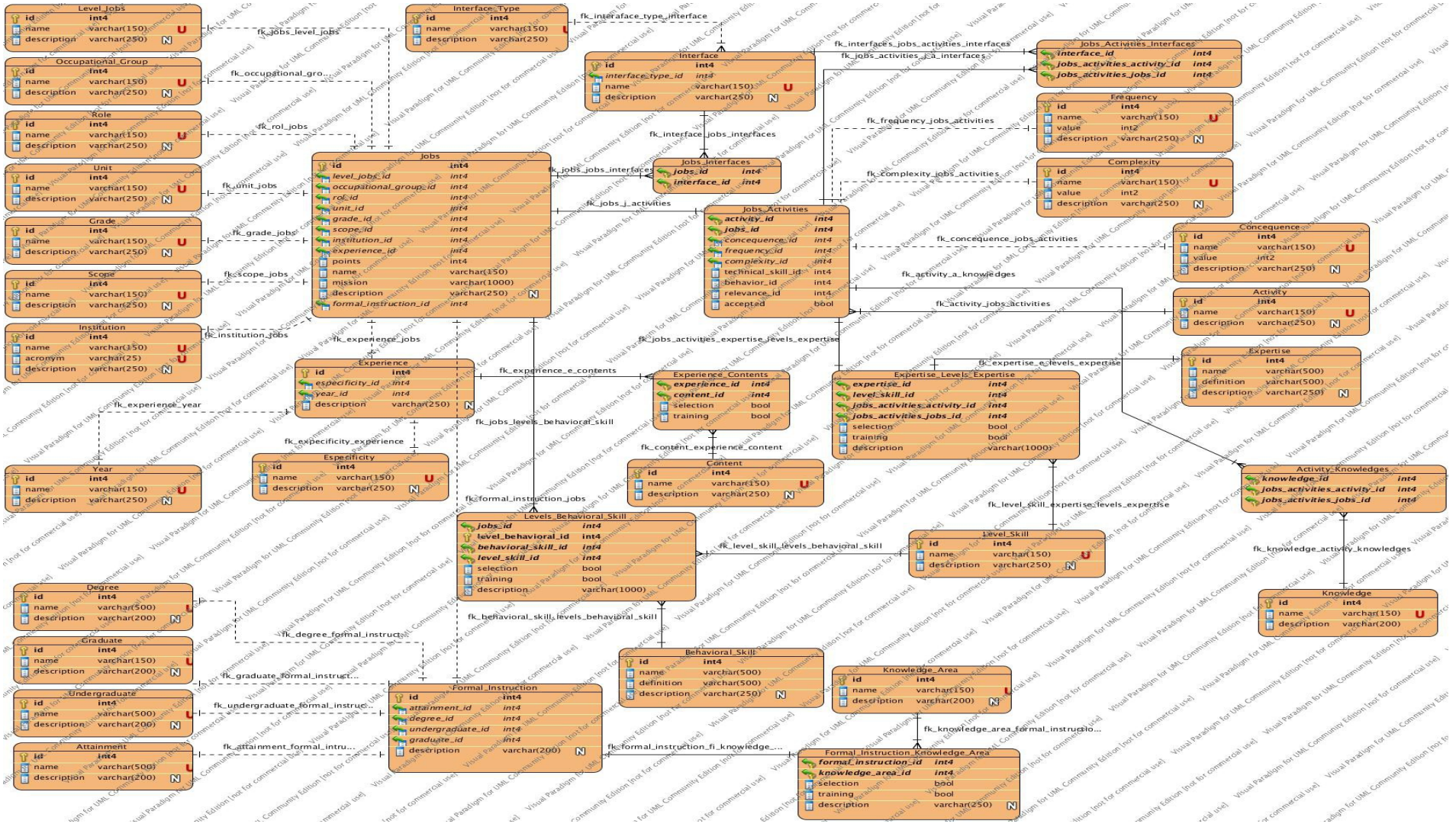


Realizado por: Diego P. Tamayo Barriga



Realizado por: Diego P. Tamayo Barriga

D. Diagrama relacional de la Base de Datos

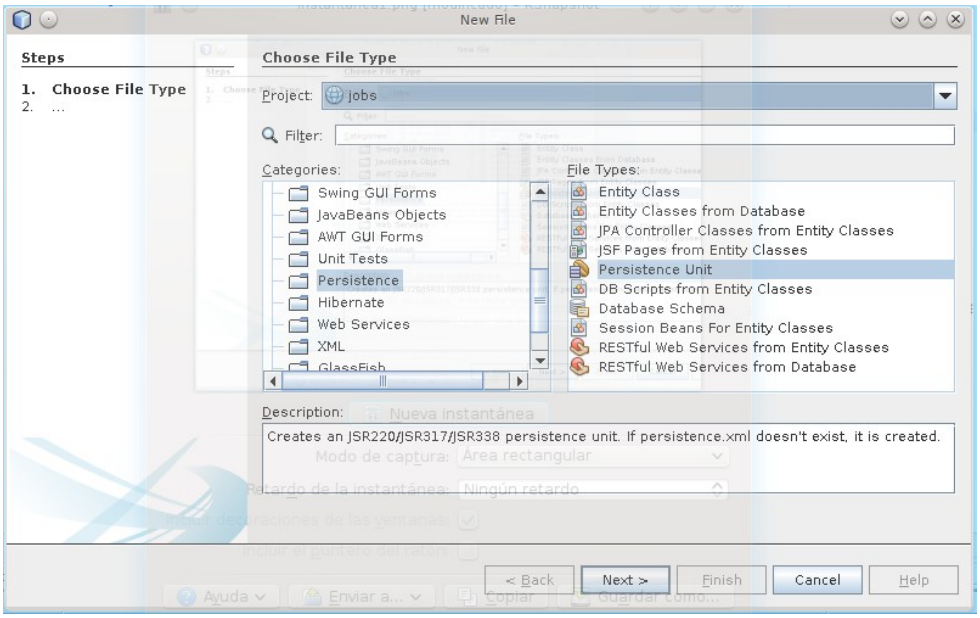


Realizado por: Diego P. Tamayo Barriga

E. Capa de Persistencia (modelo)

Esta tarea se ejecutó con Netbeans y la base de datos creada, para lo cual los pasos a seguir son:

1. Previamente se debe crear un proyecto web.
2. [File]->[New File]->[Categories -> Persistence]->[File Type -> Persistence Unit] y [Next]

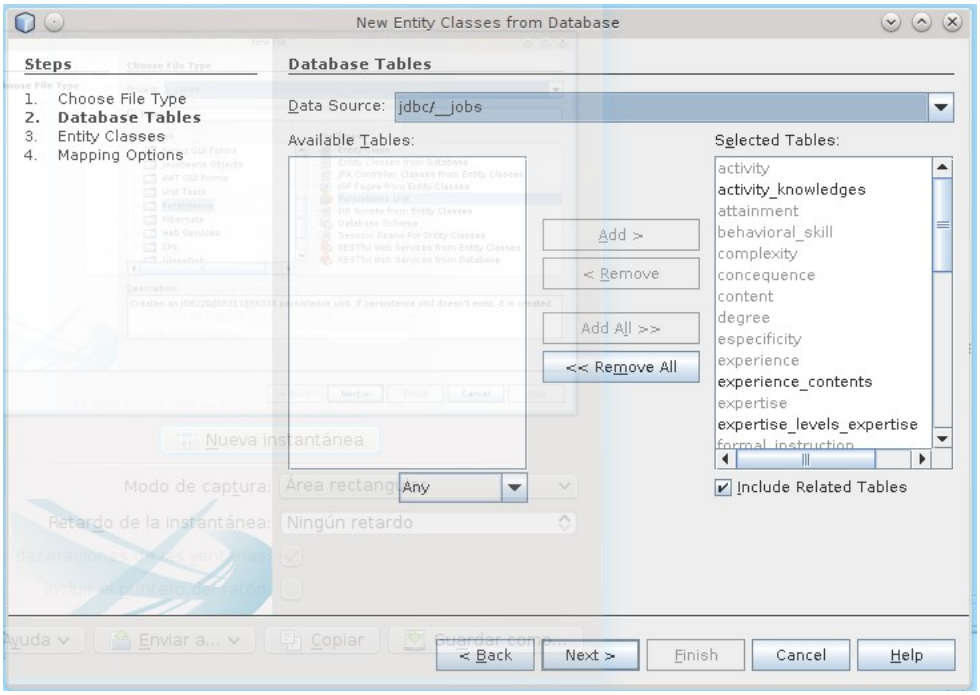


Realizado por: Diego P. Tamayo Barriga

Persistence Unit Name: jobsPU

Data Source: jdbc/__jobs

3. [File]->[New File]->[Categories -> Persistence]->[File Type -> Entity Classes From Database] y [Next]

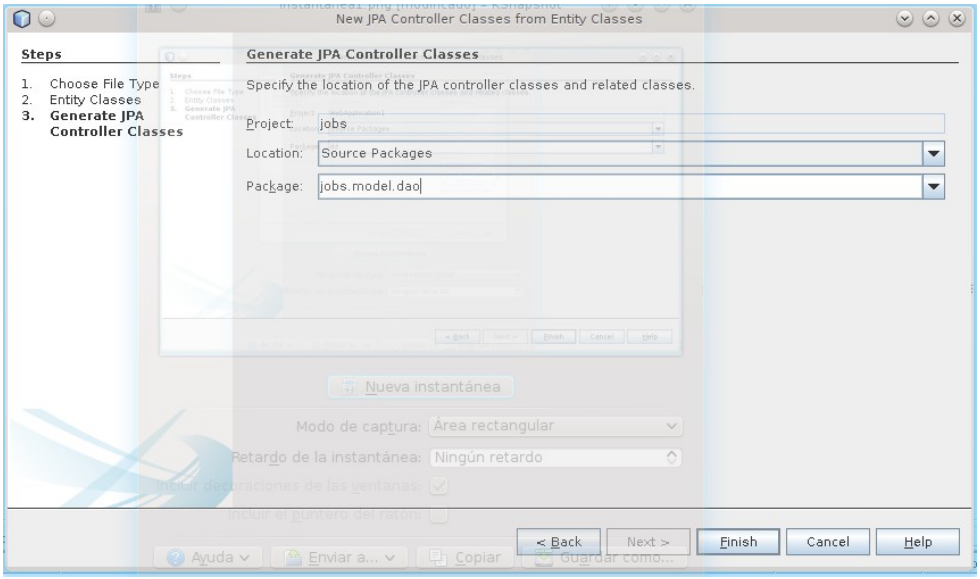


Realizado por: Diego P. Tamayo Barriga

Ventana [Entity Clases] y [Finish]

Package: jobs.model

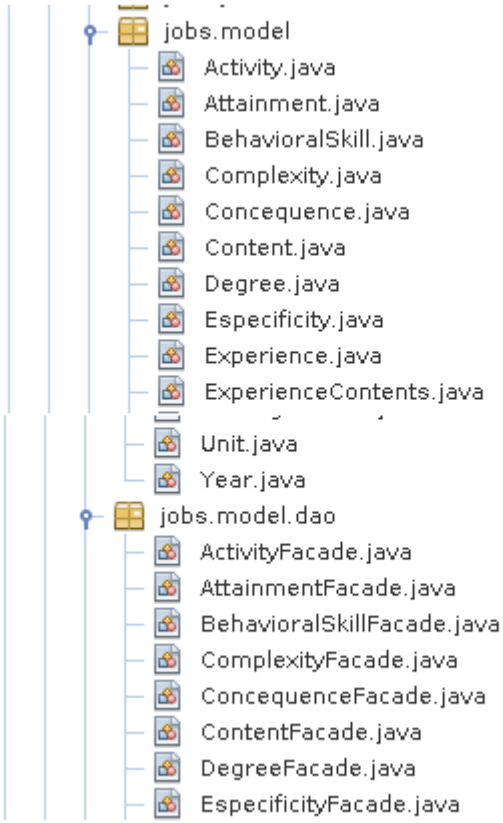
4. [File]->[New File]->[Categories -> Persistence]->[File Type -> JPA Controllers Classes from Entity Classes] y [Next]. Se debe seleccionar las clases con las que se necesite desde [Available Entity Classes] a [Selected Entity Classes] y [Next]



Realizado por: Diego P. Tamayo Barriga

Package: jobs.model.dao

5. En nuestro proyecto se creó un paquete con el nombre especificado en el paso anterior con la lógica para la creación, lectura, actualización y eliminación de nuestras entidades de clases.

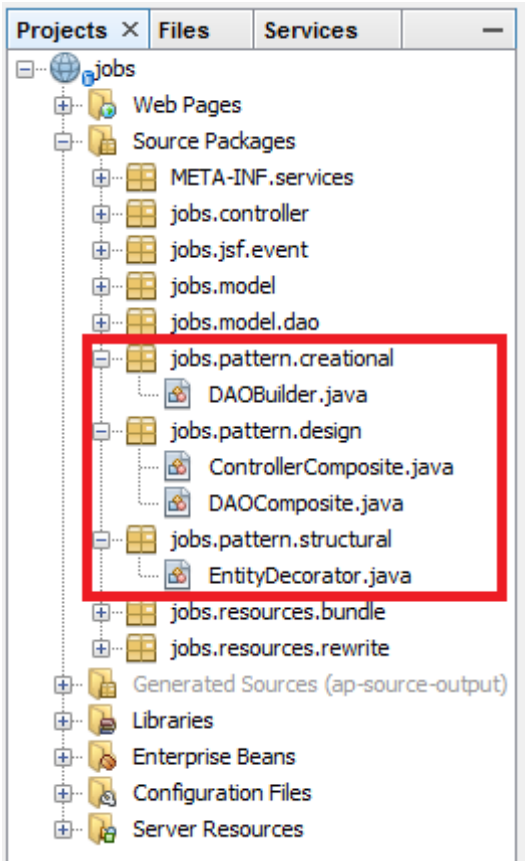


Realizado por: Diego P. Tamayo Barriga

F. Patrones de diseño en código

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño resulta ser una solución a un problema anterior. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares. Otra es que debe ser reutilizable, etc.



Realizado por: Diego P. Tamayo Barriga

Para el desarrollo del proyecto se tomaron en cuenta algunos de estos patrones, los cuales se les puede ver en la siguiente figura dentro del Sistema

para Recursos Humanos.

Patrón de Diseño - Build

Este permite tener una interfaz común para la creación de entidades y su almacenamiento en base de datos.

```java
/**
 * Interfaz abstracta para el almacenamiento de objetos en base de datos
 *
 * @author Paul Tamayo
 * @version 1.0
 */
public interface DAOBuilder<T> {

    /**
     * Permite almacenar un objeto creado
     *
     * @param dao
     */
    void create(T dao);

    /**
     * Permite almacenar un objeto modificado
     *
     * @param dao
     */
    void edit(T dao);

    /**
     * Permite eliminar un objeto
     *
     * @param dao
     */
    void remove(T dao);
```

Realizado por: Diego P. Tamayo Barriga

Está permite construir objetos de entidades sin conocer la instancia final de la entidad.

```java
public abstract class DAOComposite<T> implements DAOBuilder<T> {

    private Class<T> entityClass;

    /**
     * Constructor que permite crear un objeto que permite construir y almacena
     * en base de datos
     *
     * @param entityClass
     */
    public DAOComposite(Class<T> entityClass) {
        this.entityClass = entityClass;
    }

    protected abstract EntityManager getEntityManager();

    @Override
    public void create(T entity) {
        getEntityManager().persist(entity);
    }

    @Override
    public void edit(T entity) {
        getEntityManager().merge(entity);
```

Realizado por: Diego P. Tamayo Barriga

Patrón de Diseño - Decorator

Permite definir atributos comunes para las entidades para su posterior uso.

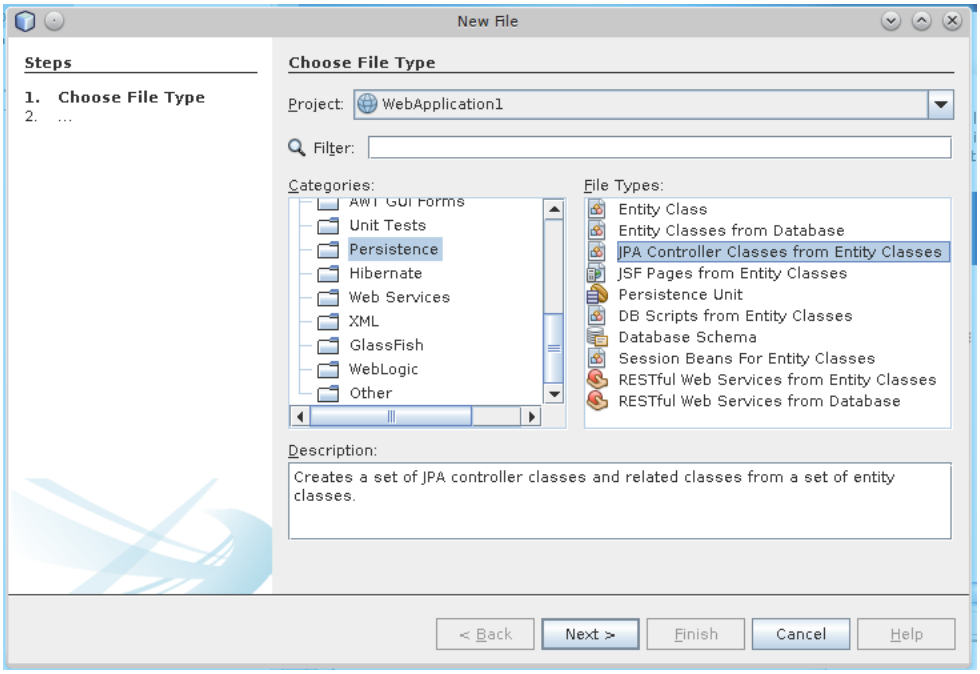
```java
/**
 * Decorador para atributos comunes para las entidades
 *
 * @author Paul Tamayo
 */
public interface EntityDecorator {

    /**
     * Nos permite devolver el identificador de la entidad.
     *
     * @return id
     */
    Integer getId();

    /**
     * Nos permite modificar el identificador de la entidad.
     *
     * @param id nueva identidad
     */
    void setId(Integer id);

    /**
     * Devuelve el nombre de la entidad
     *
     * @return nombre de la entidad
     */
    String getEntityName();
```

Realizado por: Diego P. Tamayo Barriga

G. Controlador de entidades

Esta tarea se ejecutó con Netbeans y la base de datos creada, para lo cual los pasos a seguir son:

1. [File]->[New File]->[Persistence]->[JPA Controller Classes from Entity Classes]->[Next]



Realizado por: Diego P. Tamayo Barriga

2. En la ventana de [Entity Classes] seleccionamos las clases las cuales se quiera generar su controlador y [Next].

3. En la ventana [Generate JPA Controller Classes], en esta ventana en la opción [Package] se añade un nombre el cual en el caso de este proyecto va a ser jobs.controller y [Finish]

4. Así ya se tiene creado los controladores para el proyecto.

H. Creación de las plantillas en JavaServer Faces

Ahora que se conoce las principales novedades con que cuenta JavaServer Faces, con la ayuda del framework JavaServer Faces también se cuenta con un poderoso conjunto de etiquetas para un mejor control de plantillas. Este framework cuenta con muchas características siendo las más importantes:

- Tiempo de desarrollo cero de las etiquetas para UIComponents.
- Facilidad en la creación de las plantillas para los componentes y páginas.
- Habilidad de separar los UIComponents en diferentes archivos.
- Un buen sistema de reportes de errores.
- Validación en tiempo de construcción.

Etiqueta	Descripción
ui:composition	Envuelve un conjunto de componentes para ser reutilizados en otras páginas, es la etiqueta que: • Envuelve o puede envolver la plantillas • Se utiliza para hacer referencia a una plantilla. Todo lo que quede fuera de la etiqueta, no será renderizado.
ui:define	Define un contenido nombrado para ser insertado en una plantilla, su contenido es un conjunto de componentes y se identifica con un name. Ese conjunto de componentes será insertado en una etiqueta ui:insert con el mismo nombre.
ui:insert	Declara un contenido nombrado que debe ser definido en otra página.
ui:decorate	Es la etiqueta que sirve para hacer referencia a una plantilla, como la etiqueta ui:composition, solo que con ui:decorate lo definido antes y después de la etiqueta si será renderizado.
ui:param	Nos sirve para declarar parámetros y su valor en el uso de plantillas y componentes por composición.
ui:include	Es una etiqueta que sirve para incluir en una página el

	contenido de otra, como si el contenido de está última formase parte de la primera.

Diseño de plantilla pantalla principal

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:p="http://primefaces.org/ui">
  <h:head>
    <title>
      <ui:insert name="title">#{messages.appTitle}</ui:insert>
    </title>
    <link rel="icon" type="image/png" href="#{resource['images:favicon.png']}" />
    <h:outputScript library="js" name="validation.js" />
    <h:outputStylesheet library="css" name="principal.css" />
  </h:head>
  <h:body>
    <div class="container">
      <img alt="Departamento de Recursos Humanos" src="#{resource['images:rrhh.gif']}" style="height: 150px; width: 100%;" />
      <ui:insert name="toolbar">
        <div class="navigation">
          <ul class="nav menu nav-pills">
            <li class="active">
              <h:link outcome="/index" value="Inicio"/>
            </li>
            <li>
              <h:link outcome="/catalog/list" value="Catálogo"/>
            </li>
            <li>
              <h:link outcome="/description/index" value="Descripción de Puestos" />
            </li>
          </ul>
        </div>
      </ui:insert>
      <div class="content">
        <ui:insert name="content">
          Contenido
        </ui:insert>
      </div>
    </div>
    <div class="footer">
      <div class="container-footer">
        <h:form>
          <hr/>
          <p class="pull-right">
            <p:commandLink id="about" styleClass="a-foot" value="About" onclick="dlg.show();" type="button"/>
          </p>
          <p>
            © Experiencia de un Desarrollador 2013
          </p>
          <p:dialog widgetVar="dlg" header="Acerca de..." showEffect="explode" modal="true" hideEffect="bounce" height="80" width="310">
            <table>
              <tr>
                <td style="font-weight: bold;">Relizado por:</td>
                <td>Diego Tamayo<br/></td>
              </tr>
              <tr>
                <td style="font-weight: bold;">Email:</td>
                <td><a href="mailto:dpaul.tamayo@outlook.com">dpaul.tamayo@outlook.com</a></td>
              </tr>
              <tr>
                <td colspan="2" style="font-weight: bold; text-align: center;">Riobamba - Ecuador</td>
              </tr>
            </table>
          </p:dialog>
        </h:form>
      </div>
    </div>
  </h:body>
</html>
```

Realizado por: Diego P. Tamayo Barriga

Diseño de la pantalla de catálogos

```
1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE composition
3   PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
4   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
5 <ui:composition xmlns="http://www.w3.org/1999/xhtml"
6             xmlns:ui="http://java.sun.com/jsf/facelets"
7             template="./principal_layout.xhtml"
8             xmlns:p="http://primefaces.org/ui"
9             xmlns:h="http://java.sun.com/jsf/html"
10             xmlns:f="http://java.sun.com/jsf/core">
11   <ui:define name="title">
12     #{messages.appTitleCatalog}
13   </ui:define>
14   <ui:define name="toolbar">
15     <div class="navigation">
16       <ul class="nav menu nav-pills">
17         <li>
18           <h:link outcome="/index" value="Inicio"/>
19         </li>
20         <li class="active">
21           <h:link outcome="/catalog/list" value="Catálogo"/>
22         </li>
23         <li>
24           <h:link outcome="/description/index" value="Descripción de Puestos" />
25         </li>
26       </ul>
27     </div>
28   </ui:define>
29 </ui:composition>
```

Realizado por: Diego P. Tamayo Barriga

I. Planificación del SCRUM

Introducción

Este documento describe la implementación de la metodología de trabajo Scrum en el Departamento de Sistemas y Telecomunicación de la Escuela Superior Politécnica de Chimborazo, para la gestión del desarrollo del proyecto de Módulos de Catálogos del Sistema de Recurso Humano.

Incluye junto con la descripción de este ciclo de vida iterativo e incremental para el proyecto, los artefactos o documentos con los que se gestionan las tareas, seguimiento de avances, así como las responsabilidades y compromisos de los participantes en el proyecto.

Propósito de este documento

Facilitar la información de referencia necesaria a las personas implicadas en el desarrollo del sistema.

Descripción General de la metodología

Fundamentación

Las principales razones del uso de un ciclo de desarrollo iterativo e incremental de tipo Scrum para la ejecución de este proyecto son:

- Sistema modular. Las características del módulo de catálogos del Sistema de Recursos Humanos permiten desarrollar una base funcional mínima y sobre ella ir incrementando las funcionalidades o modificando el comportamiento.
- Entregas frecuentes y continuas al cliente de los módulos terminados, de forma que puede disponer de una funcionalidad

básica en un tiempo mínimo y a partir de ahí un incremento y mejora continua del sistema.

- Previsible inestabilidad de requisitos:
  - Es posible que el sistema incorpore más funcionalidades de las inicialmente identificadas.
  - Es posible que durante la ejecución del proyecto se altere el orden en el que se desean recibir los módulos o historias de usuario terminadas.
  - Para el cliente resulta difícil precisar cuál será la dimensión completa del sistema y su crecimiento puede continuarse en el tiempo, suspender o detenerse.

Valor de trabajo

Los valores que deben ser practicados por todos los miembros involucrados en el desarrollo y aunque hacen posible que la metodología Scrum tenga éxito son:

- Autonomía del equipo
- Respeto en el equipo
- Responsabilidad y auto-discriplina
- Foco en la tarea
- Información transparencia y visibilidad.

Personas y roles del proyecto

Persona	Contacto	Rol
Ing. Edgar Carga	eecarguay@yahoo.es	Product Owner
Ing. Gustavo Hidalgo	ghidalgo@espoch.edu.ec	Scrum Manager/Equipo

		Desarrollo
Diego Tamayo	dtamayo@espoch.edu.ec	Equipo Desarrollo

Artefactos

Documentos

- Product Backlog
- Sprint Backlog

Sprint

Producto Backlog

Es equivalente a los requisitos del sistema o del usuario en esta metodología.

El Product Owner puede recabar las consultas y asesoramiento que pueda necesitar para su redacción y gestión durante el proyecto al Scrum Manager de este proyecto.

Responsabilidades del gestor de producto

- Registró en la lista de pila del producto de las historias de usuario que definen el sistema.
- Mantenimiento actualizado de la pila del producto en todo momento durante la ejecución del proyecto.
    - Orden en el que desea quiere recibir terminada cada historia de usuario.
    - Incorporación / eliminación /modificaciones de las historias o de su orden de prioridad.

Responsabilidades del Scrum Manager

- Supervisión de la pila de producto, y comunicación con el Product Owner para pedirle aclaración de las dudas que pueda tener, o asesorarle para la subsanación de las deficiencias que observe.

Responsabilidades del equipo:

- Conocimiento y comprensión actualizada de la pila del producto.
- Resolución de dudas o comunicación de sugerencias.

Nº	Descripción	Prioridad
1	Maquetación de Pantalla Principales y Secundarias	1
2	Creación de las hojas de estilo para las distintas pantallas	4
3	Creación de las plantillas JavaServer Faces para las distintas pantallas	3
4	Creación del diagrama relacional para la base de datos	1
5	Creación de la base de datos	2
6	Creación de la capa de persistencia (Modelo)	2
7	Definir entidades que corresponden al módulo de catálogos	3
8	Creación de los controladores de entidades del módulo de catálogos (Controlador)	3
9	Creación de las vistas de las entidades del módulo de catálogos	3
10	Creación del sistema de escritura de URL	4

Pila del sprint

Es el documento de registro de los requisitos detallados o tareas que va a desarrollar el equipo en la iteración actual.

Responsabilidades del gestor de producto

- Presencia en las reuniones en las que el equipo elabora la pila del sprint.

- Resolución de dudas sobre las historias de usuario que se descomponen en la pila del sprint.

Responsabilidades del Scrum Manager

- Supervisión y asesoría en la elaboración de la pila del sprint.
- Responsabilidades del equipo técnico

Elaboración de la pila del sprint.

- Resolución de dudas o comunicación de sugerencias sobre las historias de usuario con el gestor del producto.

Sprint

Cada una de las iteraciones del ciclo de vida iterativo Scrum así como la duración de cada sprint.

Sprint 0 o Análisis previo

En este sprint tuvo aproximadamente la duración de unos 15 días, en los que se realizó el levantamiento de los requerimientos.

Nº	Tarea	Tipo
1	Maquetación de Pantalla Principales y Secundarias	Análisis/Diseño
2	Creación del diagrama relacional para la base de datos	Análisis/Diseño

Sprint 1

En este sprint tuvo aproximadamente la duración de unos 15 días, este se centró en integrar la parte de la base de datos para la posterior utilización en las diferentes tareas.

Nº	Tarea	Tipo
1	Creación de la base de datos	Desarrollo
2	Creación de la capa de persistencia (Modelo)	Desarrollo

3	Creación de las plantillas JavaServer Faces para las distintas pantallas	Desarrollo
4	Definir entidades que corresponden al módulo de catálogos	Desarrollo

Sprint 2

En este sprint tuvo aproximadamente la duración de unos 15 días, está se centra en el desarrollo del módulo de catálogos del Departamento de Recursos Humanos.

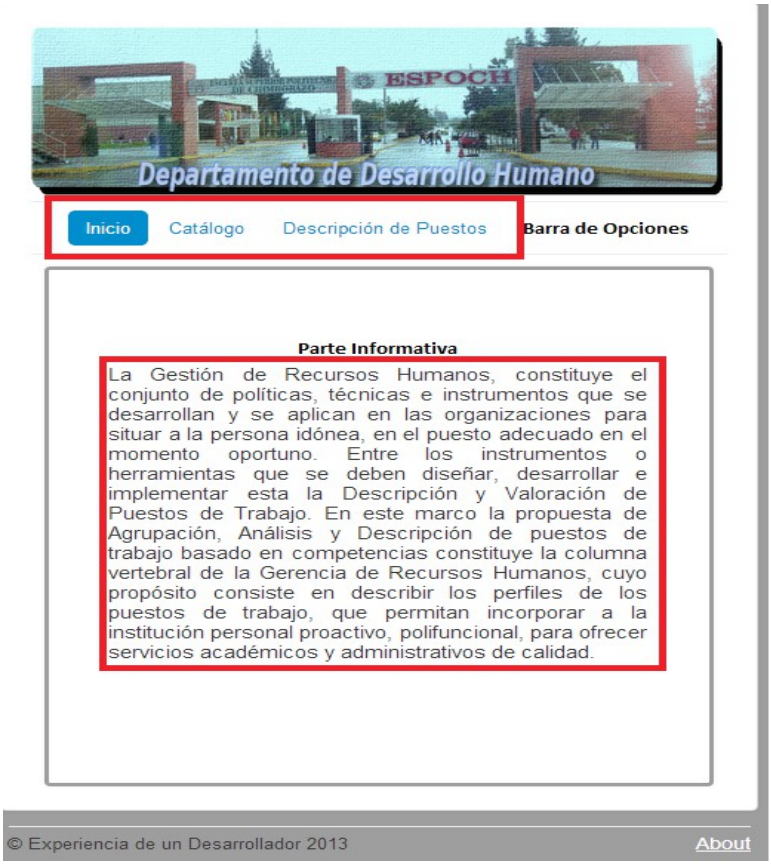
Nº	**Tarea**	**Tipo**
1	Creación de los controladores de entidades del módulo de catálogos (Controlador)	Desarrollo
2	Creación de las vistas de las entidades del módulo de catálogos	Desarrollo
3	Creación del sistema de escritura de URL	Desarrollo

J. Manual de Usuario

El presente documento tiene como objetivo explicar de manera clara el funcionamiento del sistema de módulo de Catálogos del sistema de Recursos Humanos, repasando los puntos más importantes del proceso en el sitio.

Pantalla Principal

Al ingresar al sistema, se dirige a la pantalla principal del sistema en el cual se encuentra una barra de opciones y la información del departamento de Recursos Humanos.



Realizado por: Diego P. Tamayo Barriga

Administración de Catálogos

Para ingresar a la administración de catálogos, se debe presionar dentro de la barra de opciones la opción de catálogos.

Realizado por: Diego P. Tamayo Barriga

Dentro de este se puede observar que se tiene agrupado a los catálogos de acuerdo a una necesidad dentro del sistema de Recursos Humanos.

Esto permite un mejor manejo de la información y así reducir al máximo la inconsistencia en base de datos.

Catálogo General

Muestra una tabla de opciones los cuales permiten ingresar en base de datos cada uno de estas entidades.

Realizado por: Diego P. Tamayo Barriga

Catálogo de Instrucción Formal

Muestra una tabla de opciones los cuales permiten ingresar en base de datos la información para completar una instrucción formal.

Realizado por: Diego P. Tamayo Barriga

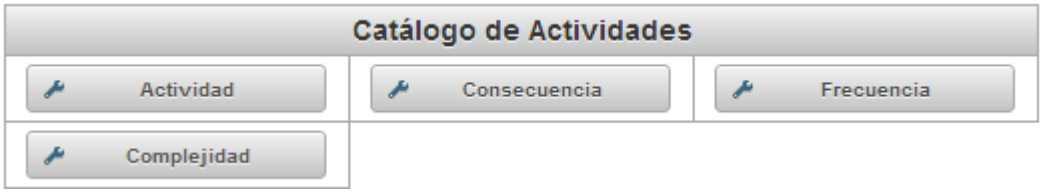
Catálogo de Experiencia

Muestra una tabla de opciones los cuales permiten ingresar en base de datos la información para completar la experiencia.

Realizado por: Diego P. Tamayo Barriga

Catálogo de Actividades

Muestra una tabla de opciones los cuales permiten ingresar en base de datos la información para completar una actividad.



Realizado por: Diego P. Tamayo Barriga

Catálogo de Destreza

Muestra una tabla de opciones los cuales permiten ingresar en base de datos la información para completar una destreza.

Realizado por: Diego P. Tamayo Barriga

Para el funcionamiento de las distintas opciones de los diferentes catálogos, existe una pantalla en común para lo cual se procede a explicar una de ellas y las demás serán similares a la descrita. Para la demostración se tomará la opción de Títulos del Catálogo de Instrucción Formal.

Para poder ingresar a la pantalla principal de Títulos basta con dar un clic en el botón y automáticamente se dirige a la pantalla de Administración de Catálogos – Títulos.

Realizado por: Diego P. Tamayo Barriga

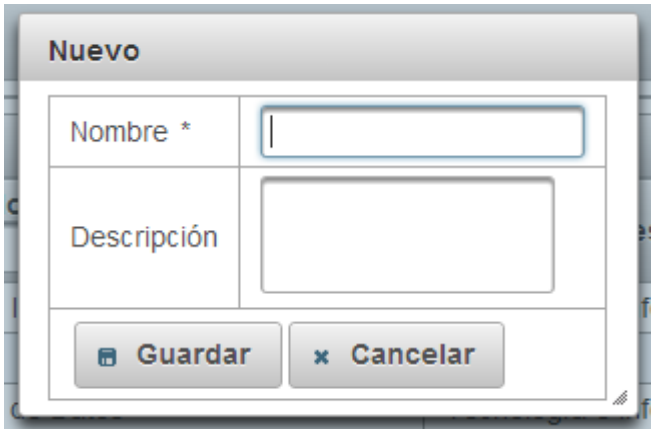
En la imagen se puede verlas diferentes partes que la componen.

1. **Barra de Títulos:** En esta se presenta el nombre del catálogo que se está registrando la información.
2. **Barra de Herramientas:** En esta se dispone de botones con las opciones de nuevo, editar y eliminar un Título.
3. **Filtros de Búsqueda:** Permite realizar búsqueda de los títulos registrados en base de datos.
4. **Paginación:** Permite desplazarnos entre las diferentes páginas de los títulos registrados.

Para crear un nuevo título basta con dar clic en el botón respectivo de la barra de herramientas:

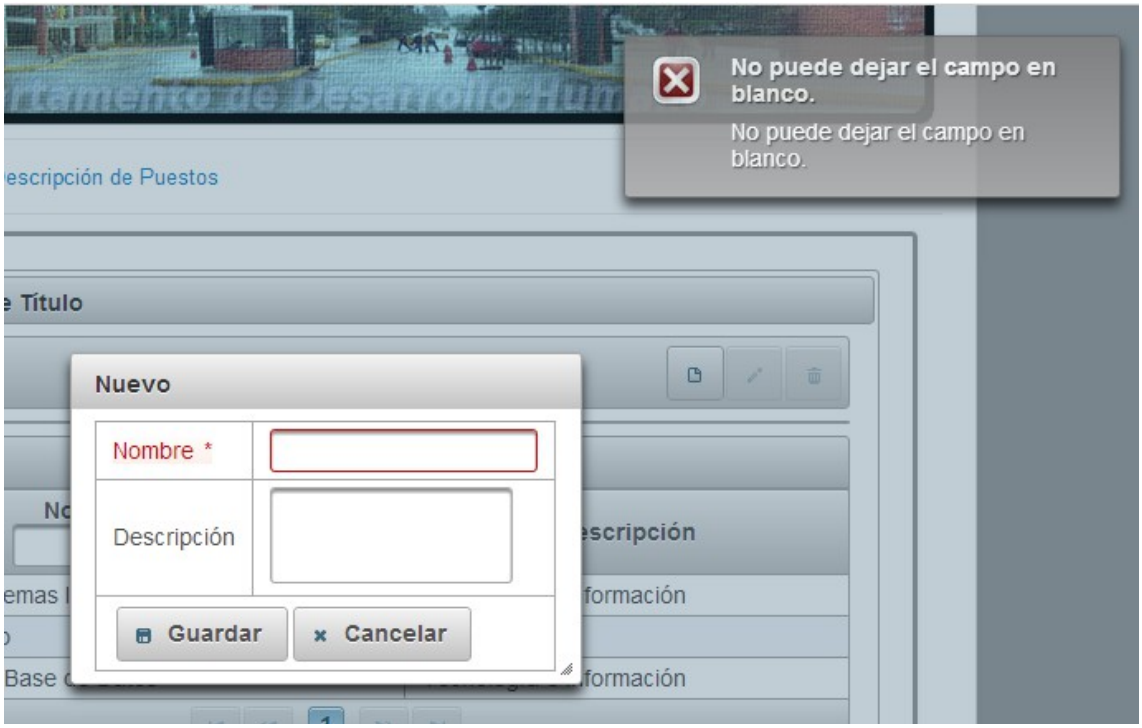
Realizado por: Diego P. Tamayo Barriga

Se abrirá un cuadro de dialogo donde se podrá ingresar la información para el registro de un nuevo título.



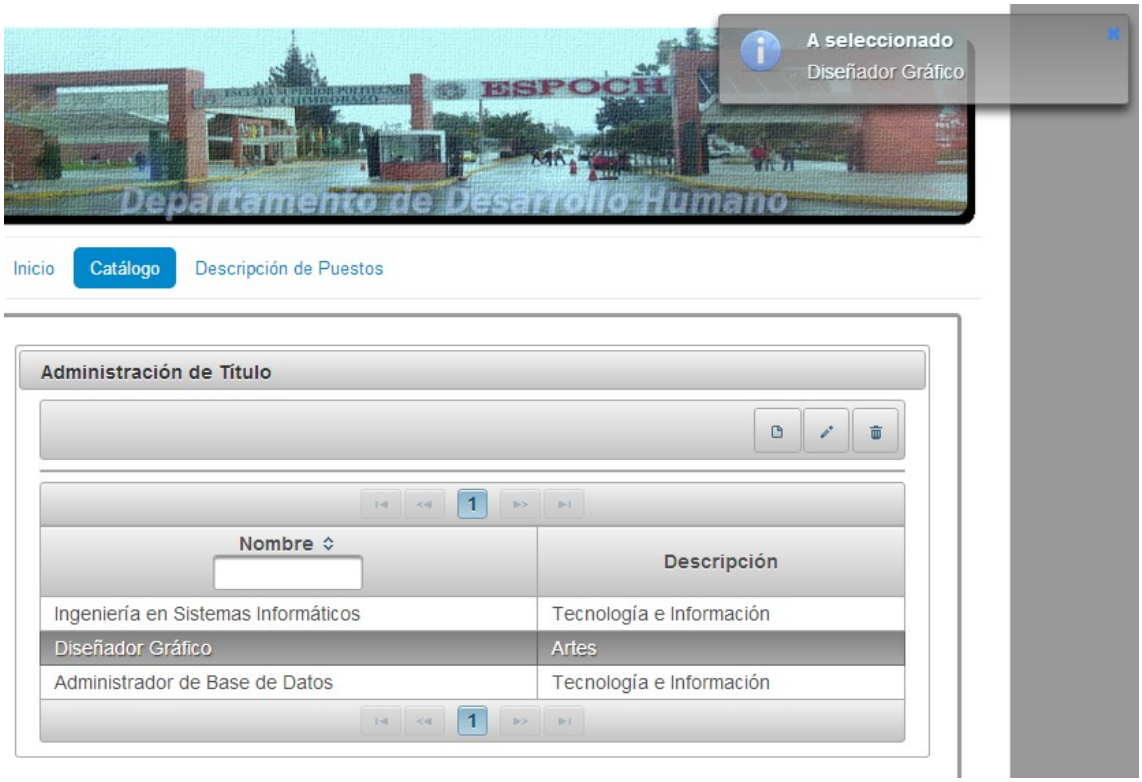
Realizado por: Diego P. Tamayo Barriga

Al cometer un error en el ingreso de datos se presenta la información y se pinta el recuadro.



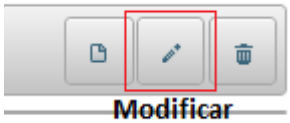
Realizado por: Diego P. Tamayo Barriga

Para modificar un registro de un título que se haya ingresado con información incorrecta basta con dar clic en el nombre del título a modificar. Y se procede como si fuese un registro nuevo, con la única novedad que la información ha sido cargada en los diferentes componentes.



Realizado por: Diego P. Tamayo Barriga

Después se habilitara dentro del cuadro de herramientas las opciones para modificar y eliminar.

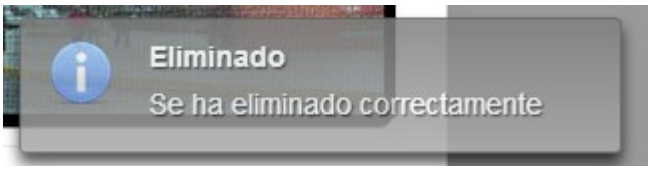


Realizado por: Diego P. Tamayo Barriga

Al igual que modificar el momento de seleccionar un título se habilita el botón de eliminar.

Realizado por:
Diego P. Tamayo Barriga

Hay que tener cuidado porque una vez dado clic se procede a borrar y no puede ser recuperado. Este mostrará un mensaje el cual confirma su eliminación.



Realizado por: Diego P. Tamayo Barriga

Descripción de puestos de trabajo

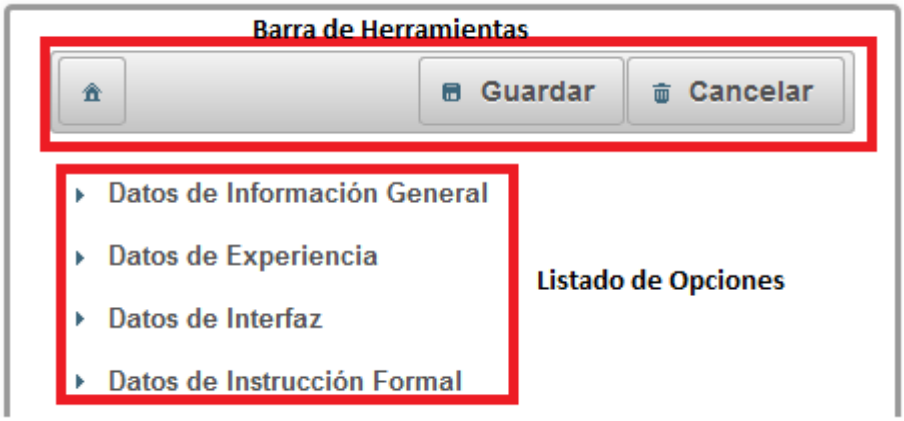
Al igual que la administración de Catálogos también se dispone de una ventana para Descripción de Puestos de Trabajo, consta de las mismas partes en estructura.

Realizado por: Diego P. Tamayo Barriga

Para el ingreso de un nuevo Puesto de Trabajo se realizara al igual que las anteriores bastaría con dar clic en Nuevo.

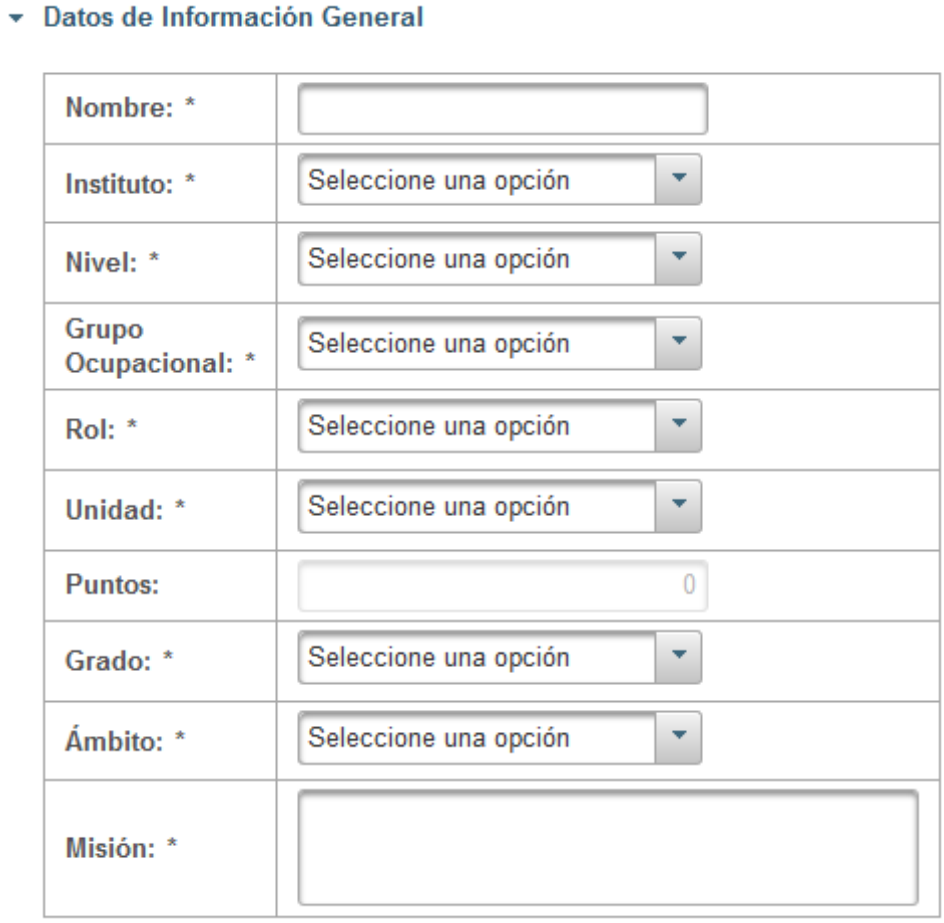
Se abrirá una ventana el cual consta de las siguientes partes:

- **Barra de Herramientas:** En esta sección se tiene la opción de Guardar y Cancelar al lado derecho de la barra y al lado izquierdo un botón para regresar a la pantalla principal.
- **Listado de Opciones:** Entre las opciones con las que cuenta son:
    - Datos de Información General
    - Datos de Experiencia
    - Datos de Interfaz
    - Datos de Instrucción Formal



Realizado por: Diego P. Tamayo Barriga

Datos de Información General

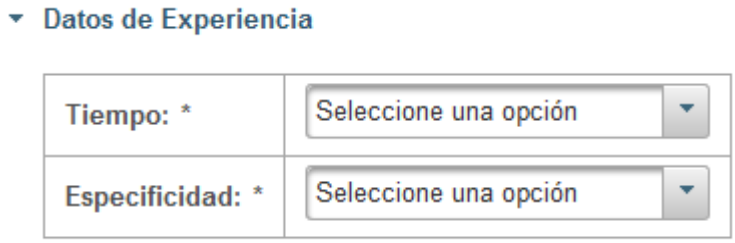


Realizado por: Diego P. Tamayo Barriga

En esta sección contiene información general del puesto.

Datos de Experiencia

En esta sección se muestra la experiencia que un postulante debe tener para aplicar al puesto de trabajo.



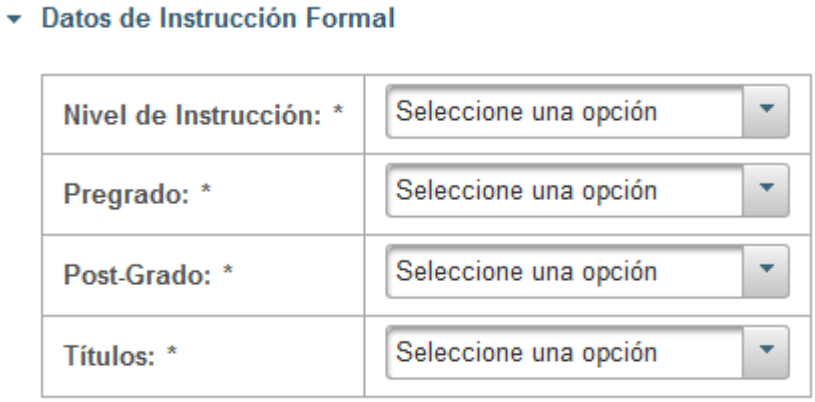
Realizado por: Diego P. Tamayo Barriga

Datos de Interfaz

Realizado por: Diego P. Tamayo Barriga

Datos de Instrucción Formal

En esta sección se ingresa el nivel de instrucción que necesita un postulante para el puesto de trabajo.



Realizado por: Diego P. Tamayo Barriga

Al terminar de ingresar la información en cada una de las secciones de la descripción de puestos de trabajo se puede guardar la información pulsando el botón Guardar de la Barra de Herramienta.

La aplicación móvil es muy amigable al cometer algún error nos mostrara un mensaje y pinta el campo en el que se está cometiendo dicho error.