Diseño e implementación de un componente gráfico para el Framework de presentación JavaServer Faces

José Luis García Deza

II

Josep María Camps Riba

25 de junio de 2008

A mi querida mujer y mis pequeñas, por estar siempre ahí y comprender en todo momento el tiempo que hemos sacrificado durante estos tres años de estudio.

Resumen

El presente proyecto consiste en el estudio y evaluación de algunos de los marcos de trabajo más utilizados para el desarrollo de aplicaciones JEE, con el fin de seleccionar uno de ellos y ampliarlo. Añadiéndole un componente gráfico que facilite, al desarrollador, la tarea de realizar una representación gráfica de un conjunto de datos.

El trabajo se presenta dividido en tres partes bien diferenciadas:

- Estudio y evaluación de los Frameworks de presentación.
- Análisis y Diseño del Componente gráfico.
- Desarrollo de una aplicación web de ejemplo.

En la primera parte, se realiza un estudio pormenorizado de tres de los Frameworks de presentación más utilizados: Struts, Spring y JavaServer Faces (JSF). Además de la definición, se han incluido los aspectos más destacables y los puntos débiles de cada uno de los Frameworks, lo que nos permitirá evaluarlos y poder realizar una comparativa que nos ayude a seleccionar el Framework sobre el que se implementará nuestro componente.

Ya en la segunda parte, se realiza el análisis, diseño e implementación del componente gráfico, que permitirá mostrar una representación gráfica en formato Adobe Flash de un conjunto de datos. El componente permitirá representar los datos suministrados en los formatos más habituales: Columnas, Líneas, Barras 3D, Áreas, Circular, Dispersión y Cotizaciones. Igualmente, se podrán parametrizar una gran cantidad de propiedades como: tamaño de la película Flash, título de la gráfica, leyendas de los ejes X e Y, colores de fondo y de línea, etc.

Por último, se realiza el análisis, diseño e implementación parcial de una aplicación web que nos permita integrar el componente para demostrar su uso. La aplicación se encarga de mostrar las estadísticas de los recorridos de las partidas de los jugadores de golf.

Summary

The aim of this project is to examine and assess some of the most frequently used frameworks in the development of JEE applications. Its purpose is to select and expand one of them by adding to it a graphic component which facilitates the developer the task of making a graphic representation of a data set.

This project is divided in three clearly differentiated parts:

- Study and assessment of the presentation frameworks
- Analysis and design of the graphic component
- Development of an application of a sample web page

In the first part there is a detailed study of three of the most commonly used presentation frameworks: Struts, Spring and JavaServer Faces (JSF). Apart from the definition, the most highlighted and weakest aspects of each Framework have been included, which will allow us to evaluate and compare them so that we can select the appropriate Framework which our component will be applied to.

In the second part, an analysis, design and implementation of the graphic component is developed, which will make it possible to display an Adobe Flash graphic representation of a data set. The component permits to represent the provided data in the most frequent formats: Columns, lines, 3D Columns, Areas, Pies, Scatters, and Stocks. Likewise it will also be possible to parameterize a great amount of properties such as: size of the Flash film, graph title, caption of the axis X and Y, background colours, etc.

Lastly, the analysis, design and partial implementation of the web application will permit the component integration in order to prove its use. The application will show the statistics of the golf players game rounds.

Índice de contenidos

Índice de figuras

Memoria

Capítulo 1: Introducción.

Descripción del PFC

En la actualidad, el aspecto y la presentación de la interfaz de usuario están tomando mayor relevancia en el proceso de diseño e implementación de las aplicaciones web. Ayudado en gran medida por la aparición de marcos de trabajo (Frameworks), que facilitan este tipo de tareas al desarrollador.

Debemos partir de la idea de que la interacción con las aplicaciones deben ser fáciles de aprender y utilizar, y por tanto intuitivas. Además deben satisfacer las necesidades de los usuarios, lo que implica que deben ser funcionales.

Estas reflexiones nos llevan a pensar en la necesidad de proporcionar a los desarrolladores con más y mejores herramientas de cara a afrontar el diseño de las interfaces de usuario. Y con esta idea partimos en el desarrollo de este Proyecto.

Pretendemos estudiar y evaluar los distintos marcos de trabajo, existentes hoy en día en el mercado, con el fin de poder ampliarlos con nuevas funcionalidades, que permitan simplificar y mejorar el desarrollo de la capa de presentación de las aplicaciones web.

El trabajo se presenta dividido en tres partes bien diferenciadas:

- Estudio y evaluación de los tres marcos de trabajo más utilizados hoy en día para el desarrollo de la capa de presentación en JEE: Struts, Spring y JSF (Java Server Faces)

  Dentro de este estudio incluiremos aspectos relativos a la definición de cada marco de trabajo, sus aspectos más destacables, sus puntos débiles y una comparativa que nos permitirá realizar la elección del marco de trabajo sobre el que diseñaremos la ampliación.

- Diseño e implementación de un componente, que permita al marco de trabajo seleccionado realizar una representación gráfica, con salida en formato flash, de un conjunto de datos suministrado, según la configuración indicada.

  El componente nos permitirá realizar las siguientes acciones:

  - Visualizar los datos en distintos formatos gráficos: líneas, barras, circular, áreas, barras 3D, dispersión y cotizaciones.
  - Especificar un título para el gráfico.
  - Seleccionar los colores de textos y fondos.
  - Definir nombres de las categorías de datos a representar.
  - Especificar el tamaño (altura y anchura) de la película (swf) que contendrá la representación gráfica.
  - Además de indicar una gran cantidad de propiedades adicionales.

- Por último, realizaremos el análisis, diseño e implementación parcial de una aplicación web, que utilizando el marco de trabajo elegido realice una muestra de la utilización del componente desarrollado anteriormente.

  Con el fin de conseguir una mejor eficiencia a la hora de desarrollar o modificar la aplicación web, utilizaremos el patrón de diseño MVC (Model-View-Controller). El patrón MVC define una organización independiente para los objetos de negocio (Model), la interfaz de usuario (View) y el flujo de trabajo de la aplicación (Controller).

Como resultado de nuestro proyecto obtendremos pues, además de un estudio comparativo de las distintas tecnologías utilizadas en la implementación de los marcos de trabajo de presentación, el desarrollo de un nuevo componente que complementará las prestaciones del marco de trabajo elegido y una aplicación de ejemplo que demostrará su uso.

Objetivos generales y específicos

Con la realización de este Proyecto Fin de Carrera, se pretende profundizar en el conocimiento de los siguientes aspectos:

- Marcos de Trabajo de presentación: Analizar qué son los marcos de trabajo y en especial los específicos de la capa de presentación, indicando para qué sirven y cómo se implementan. El análisis se centrará en los Frameworks existentes más utilizados en la actualidad.

  Veremos las ventajas e inconvenientes, análisis sobre el funcionamiento e implementación de los siguientes marcos de trabajo:
  - Framework Struts.
  - Framework Spring.
  - Framework JSF.

- Patrón de diseño MVC: Analizar el patrón con el fin de conocer las mejoras que aporta al desarrollo de aplicaciones web.

- Tecnología Flash: Analizar la tecnología de animación flash para la generación de gráficas, estudiar la posibilidad de configurar las animaciones mediante ficheros de configuración XML y el paso de parámetros.

- Nueva implementación: Analizar, diseñar e implementar un nuevo componente que se integrará en el marco de trabajo elegido, a partir de las conclusiones obtenidas en el estudio anterior. Pretendemos con ello conseguir facilitar al desarrollador la integración de gráficas en el diseño de aplicaciones web.

- Prueba de ejemplo: Realizar una aplicación de ejemplo, que permita mostrar el uso del nuevo componente diseñado, dentro del marco de trabajo seleccionado. Permitiéndonos mostrar el avance conseguido desde el punto de vista del desarrollador.

Hitos y temporalización

La temporalización del proyecto tiene en cuenta las entregas parciales, definidas por las tres primeras PEC's, así como las diferentes tareas que deben realizarse.

Así pues, a continuación mostramos las principales fechas que deberemos tener en cuenta en el desarrollo del proyecto:

Tareas	F. Inicio	F. Fin
Plan de Trabajo del Proyecto	28-02-08	14-03-08
Estudio del patrón MVC	15-03-08	19-03-08
Estudio y análisis de los Frameworks de Presentación	20-03-08	03-04-08
Estudio de la Tecnología Adobe Flash	04-04-08	07-04-08
Análisis del nuevo componente	08-04-08	14-04-08
Diseño del nuevo componente	15-04-08	06-05-08
Implementación del nuevo componente	07-05-08	31-05-08
Prueba de ejemplo	01-06-08	14-06-08
Realización de la Memoria del Proyecto	15-06-08	21-06-08
Realización de la presentación virtual	22-06-08	25-06-08

Tabla 1: Tareas y fechas del proyecto

Los principales hitos del proyecto son:

PEC 1: (14-03-2008) coincidiendo con la finalización del Plan de Trabajo.

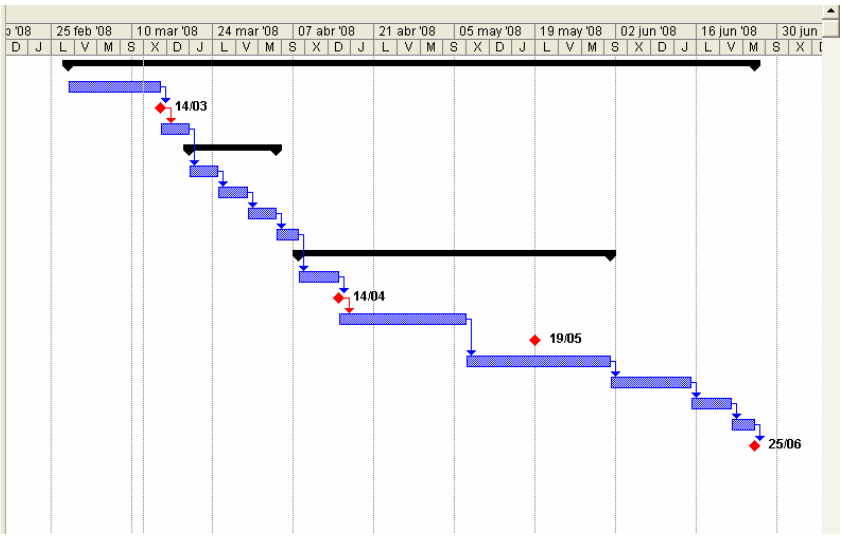
PEC 2: (14-04-2008) coincidiendo con la finalización del análisis del nuevo componente, e incluyendo el estudio de los distintos framewroks de presentación.

PEC 3: (19-05-2008) coincidiendo con la finalización del diseño del nuevo componente y ya habiendo empezado la implementación del mismo.

Entrega Final: (28-06-2008) coincidiendo con la entrega de la memoria y la presentación virtual.

Diagrama de Gantt:

	Nombre de tarea	Duración	Comienzo	Fin
1	**Proyecto Fin de Carrera**	**119 días**	**jue 28/02/08**	**mié 25/06/08**
2	Plan de trabajo	16 días	jue 28/02/08	vie 14/03/08
3	**PEC 1**	**0 días**	**vie 14/03/08**	**vie 14/03/08**
4	Estudio del patrón MVC	5 días	sáb 15/03/08	mié 19/03/08
5	**Estudio y análisis de los Frameworks de Presentación**	**15 días**	**jue 20/03/08**	**jue 03/04/08**
6	Struts Framework	5 días	jue 20/03/08	lun 24/03/08
7	Spring Framework	5 días	mar 25/03/08	sáb 29/03/08
8	JSF Framework	5 días	dom 30/03/08	jue 03/04/08
9	Estudio de la Tecnología Adobe Flash	4 días	vie 04/04/08	lun 07/04/08
10	**Nuevo Componente**	**54 días**	**mar 08/04/08**	**sáb 31/05/08**
11	Análisis del nuevo componente	7 días	mar 08/04/08	lun 14/04/08
12	**PEC 2**	**0 días**	**lun 14/04/08**	**lun 14/04/08**
13	Diseño del nuevo componente	22 días	mar 15/04/08	mar 06/05/08
14	**PEC 3**	**0 días**	**lun 19/05/08**	**lun 19/05/08**
15	Implementación del nuevo componente	25 días	mié 07/05/08	sáb 31/05/08
16	Prueba de ejemplo	14 días	dom 01/06/08	sáb 14/06/08
17	Realización de la Memoria del Proyecto	7 días	dom 15/06/08	sáb 21/06/08
18	Realización de la presentación virtual	4 días	dom 22/06/08	mié 25/06/08
19	**Entrega Final**	**0 días**	**mié 25/06/08**	**mié 25/06/08**



Productos obtenidos

Los productos que acompañan al proyecto son los siguientes:

- **Memoria**: Es el documento que está leyendo, se entrega tanto en formato Word como en PDF:
  - Jgarciadez_Memoria.doc: situado en la carpeta principal.
  - jgarciadez_Memoria.pdf: situado en la carpeta principal.
  - jgarciadez_Presentación.pps: presentación virtual. En la carpeta principal. El vídeo está en la carpeta \VideoMP.

- **Implementación del Componente: UOComponent**
  - UOComponent.jar: Librería JAR del componente diseñado. Situado en la carpeta \uocomponent\dist
  - *.java: Archivos de código fuente de la implementación. Situados en la carpeta \uocomponent\src
  - *.class: Archivos compilados de la implementación. Situados en la carpeta \uocomponent\WebRoot\WEB-INF\classes

- **Manual de uso del Componente: UOComponent**
  - UOComponent_manual.doc: situado en \uocomponent\doc
  - UOComponent_manual.pdf: situado en \uocomponent\doc

- **Implementación de la aplicación de ejemplo: GolfStat**
  - golfstat.war: WAR de la aplicación. Situado en la carpeta \golfstat\dist
  - *.java: Archivos de código fuente de la implementación. Situados en la carpeta \golfstat\src
  - *.class: Archivos compilados de la implementación. Situados en la capreta \golfstat\WebRoot\WEB-INF\classes
  - *.sql: Archivos con los Scripts de creación de la base de datos y sus tablas. Incluye la inserción de datos de ejemplo. Situados en la carpeta \golfstat\SQL.

Descripción del resto de capítulos

En el resto de capítulos de esta memoria se tratan los objetivos del proyecto, que se han descrito en el apartado Objetivos Generales y Específicos.

Capítulo 2, Conceptos previos

Pequeña introducción a conceptos previos necesarios para la comprensión del proyecto: Patrones, Frameworks y Patrón Model-View-Controller.

Capítulo 3, Estudio y comparativa de los Frameworks

Contiene el estudio y evaluación de tres de las Frameworks más utilizados en el desarrollo de aplicaciones JEE. El estudio incluye aspectos relativos a la definición de cada marco de trabajo, los aspectos más destacables y sus puntos débiles. Finalmente se realiza una comparativa de los mismos.

Capítulo 4, Análisis y diseño del componente

Describe el análisis y el diseño realizados para poder realizar la implementación del componente sobre el Framework JavaServer Faces (JSF).

Capítulo 5, Análisis y diseño de la aplicación de ejemplo

Como prueba de ejemplo, se ha diseñado una pequeña aplicación que permita utilizar el componente. En el presente capítulo se incluye el análisis y el diseño de la misma.

Capítulo 6, Conclusiones

Describe las conclusiones finales obtenidas tras realizar el proyecto.

Capítulo 2: Conceptos previos.

Patrones

Historia

Los patrones fueron ideados a finales de los años 70 por Christopher Alexander, catedrático de arquitectura de la Universidad de California, Berkeley. En sus trabajos definió el concepto de patrón y presentó un catálogo de patrones para el diseño de arquitecturas. Introduce la idea de patrón como una forma de representar la solución a problemas comunes a la mayoría de los proyectos de arquitectura.

Este concepto empezó a interesar a los desarrolladores de Programación Orientada a Objetos y fue en 1987, en la conferencia OOPSLA cuando se le empezó a dar mayor importancia a los patrones.

La mayor contribución al desarrollo de los patrones vino de la mano del grupo GoF “Gang of Four” (Banda de los Cuatro), formado por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, quienes en 1995 publicaron el libro “Design Patterns” (ISBN 0201633612), que se convirtió en la guía de referencia del diseño de patrones.

Conceptos básicos

El concepto general en el que se basan los patrones es el de ofrecer una solución experta a un problema común. Consisten pues en la estandarización de la solución al problema, por lo que se representan normalmente con plantillas.

Una plantilla básica de un patrón debería incluir: Descripción, Clasificación, Propósito, Descripción del problema, Detalle del patrón, Aplicabilidad, Implementación, Posibles variantes, Ventajas e inconvenientes, Patrones relacionados y un Ejemplo de utilización.

La aportación más importante de los patrones a la programación son la reutilización y la abstracción. Gracias a la reutilización podemos utilizar ideas y código que ya han sido usadas y probadas en proyectos previamente. La abstracción permite la división de problemas grandes en otros más pequeños y sencillos de afrontar.

Frameworks

Un framework es un conjunto de clases o componentes utilizadas conjuntamente para dar solución a un determinado problema software.

Un framework de aplicación proporciona al programador un conjunto de componentes con las siguientes características:

- Sabemos que funcionan bien, ya que han sido probados en otros proyectos.
- Pueden ser utilizados por otros desarrolladores.
- Están preparados para poderlos utilizar en los siguientes proyectos.
- Aportan una abstracción de un concepto en particular.

El principal inconveniente del uso de framework es la necesidad de llevar a cabo un proceso inicial de aprendizaje. Antes de poder reutilizar el diseño del framework, tenemos que entenderlo completamente.

Patrón Model-View-Controller

Introducción

El concepto de patrón fue descrito por primera vez en 1979 por Trygve Reenskaug, en los laboratorios de investigación de Xerox.

Modelo Vista Controlador (MVC) es un patrón que separa los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos.

Es un patrón muy utilizado en el diseño de 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 el controlador representa la Lógica de negocio.

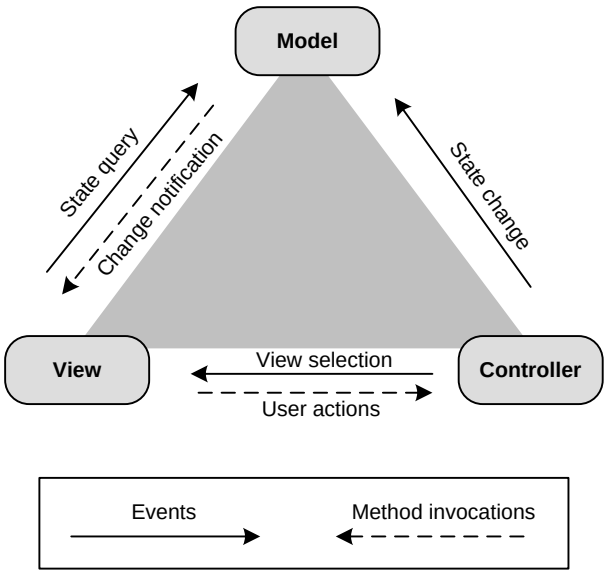


Figura 1: Flujo del patrón MVC

Son varios los frameworks JEE que utilizan MVC: Struts, Springs, Aurora, Java Server Faces y Tapestry, por ejemplo.

Plantilla

Para explicar el patrón vamos a hacerlo utilizando la plantilla que define el patrón de la que hablamos antes:

Descripción

Model-View-Controller / Modelo-Vista-Controlador

Clasificación

Patrón arquitectónico

Tipo: Comportamiento

Nivel: Componente, arquitectura.

Clasificación POSA: Interactive System

Clasificación PEAA: Presentación Web

Propósito

La idea principal consiste en desacoplar el acceso a datos y la lógica de negocio (Modelo) de la presentación de los datos y la interacción con el usuario (Vista). Para ello, MVC introduce un componente intermedio, el Controlador.

Descripción del problema

A la hora de diseñar las clases de la interfaz gráfica de usuario del sistema, debemos tener en cuanta que la interacción con el usuario es una tarea compleja y que si no se realiza una separación de responsabilidades, el mantenimiento de esta parte del sistema puede hacerse muy costoso.

Además debemos tener en cuenta que la interfaz de usuario de un sistema es, quizás, la parte más propensa a cambios.

Tratamos pues, de desacoplar la interfaz gráfica del sistema del resto del sistema.

Detalle del patrón

El patrón divide el sistema en tres tipos de componentes: modelos, vistas y controladores.

Los modelos encapsulan el estado del sistema. Estamos en el nivel de abstracción del dominio del problema. En ellos se implementa la Lógica de Negocio. Se encargan de notificar a las vistas los cambios que se producen en el estado del sistema.

Las Vistas presentan los datos a los usuarios y recogen las interacciones que se producen para enviarlas a los controladores.

Por su parte, los Controladores establecen la correspondencia entre las acciones del usuario y los acontecimientos del sistema. También deciden qué vistas se muestran a los usuarios.

Esto nos permite que cada unidad pueda ser tratada de forma independiente de las demás.

Aplicabilidad

Este patrón debemos aplicarlo en los siguientes casos:

- Separar los elementos de una aplicación en M, V y C
- Desarrollo de frameworks
    - UI como componentes
    - Vistas simultaneas para un mismo modelo
- Aplicaciones web
    - Datos XML y Representación HTML
    - No debe confundirse con 3-Capas

No se debe aplicar cuando:

- Principio de "No vas a necesitarlo":
    - Aplicar MVC es una complicación añadida.
    - Muchas aplicaciones mezclan presentación y negocio con éxito.
- Es mejor no usarlo hasta que se vea claro que los beneficios superan a las dificultades

Implementación

La implementación del patrón requiere, tal y como hemos visto antes, los siguientes componentes:

Modelo: Compuesto por una o más clases responsables de mantener los datos del modelo. El estado del modelo se mantiene en los atributos e implementación del modelo.

Vista: Las clases de la vista ofrecen una representación de los datos en el componente modelo. La vista mantiene una referencia la modelo para poder recuperar o modificar datos. Las peticiones de cambio siempre se envían al controlador.

Controlador: Gestiona los cambios en el modelo. Mantiene una referencia al modelo encargado de ejecutar los cambios, que pueden venir de la Vista.
En la siguiente figura podemos ver las relaciones entre los componentes:

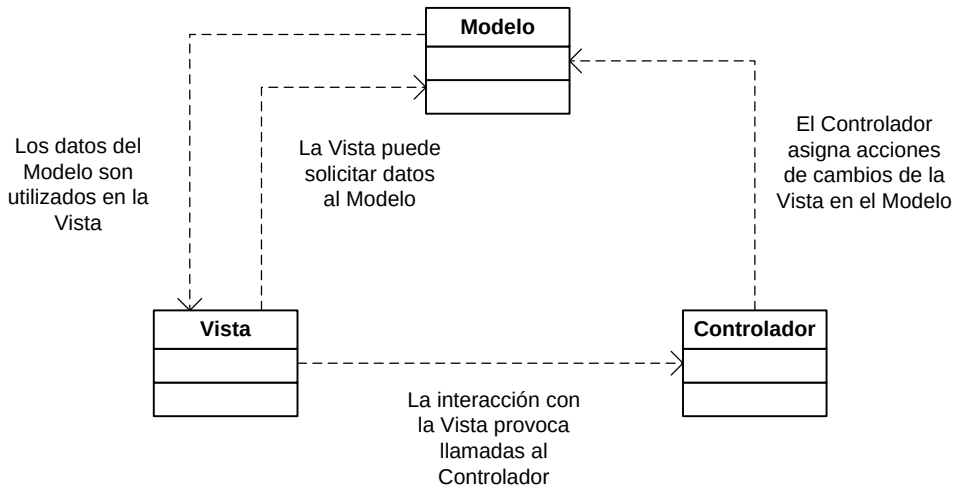


Figura 2: Relaciones componentes del patrón MVC

Posibles variantes

Ventajas e inconvenientes

En cuanto a las ventajas que ofrece el uso de MVC, podemos destacar las siguientes:

Reusabilidad, Fiabilidad, Bajos costes de desarrollo, Facilidad de mantenimiento y Rápido desarrollo de aplicaciones.

Por el contrario debemos tener en cuenta que es necesario realizar un análisis más profundo con la idea de no introducir restricciones al sistema con la aplicación del patrón.

Patrones relacionados

MVC engloba a su vez patrones verticales:

- Observer para la Vista y el Modelo.
- Composite para anidar Vistas.
- Strategy para la Vista y el Controlador.
    - Factory Method para el Controlador por defecto.
- Adapter para adaptar el Modelo a la Vista y al Controlador.

Capítulo 3: Estudio y comparativa de los Frameworks

Struts

Definición

Struts es un proyecto *open source* creado para ayudar a los desarrolladores a crear aplicaciones web de una forma rápida y sencilla. Para ello utiliza tecnologías estándar, como Java Servlets, JavaBeans y JavaServer Pages (JSP), que muchos desarrolladores suelen utilizar en sus proyectos.

Struts forma parte del Proyecto Jakarta de la Apache Software Foundation (ASF), cabe recordar que Yakarta acoge también proyectos ampliamente difundidos y utilizados entre la comunidad como Tomcat, Velocity y Ant, por ejemplo.

La primera versión de Struts fue desarrollada en la primavera de 2001, con la ayuda de más de 30 desarrolladores, su principal arquitecto y desarrollador es Craig R. McClanahan, quien también ideó Tomcat 4. Posteriormente, Craig ha sido co-líder en la especificación del marco de trabajo Java Server Faces de Sun.

Struts se basa en el patrón del Modelo-Vista-Controlador (MVC) utilizado ampliamente y de gran solidez. Tal y como hemos explicado en secciones anteriores, de acuerdo con este patrón, el procesamiento se separa en tres secciones lógicas diferenciadas, llamadas el modelo, las vistas y el controlador.

Struts proporciona al programador un conjunto de clases y tag-libs que forman el controlador, facilita la construcción de las vistas e integra la lógica de negocio en el Modelo.

Implementación del patrón MVC en Struts

Struts implementa el Modelo 2 de la Arquitectura de Sun a través de un servlet controller que puede utilizarse para controlar el flujo entre las páginas JSP y el resto de elementos de la capa de presentación.

Struts simplifica la implementación del patrón MVC, separando claramente el desarrollo de las interfaces del flujo de trabajo y la lógica de negocio. El controlador está implementado en Struts, aunque si es necesario puede heredarse, ampliarse o modificarse. El flujo de trabajo del sistema puede configurarse mediante el uso de archivos XML.

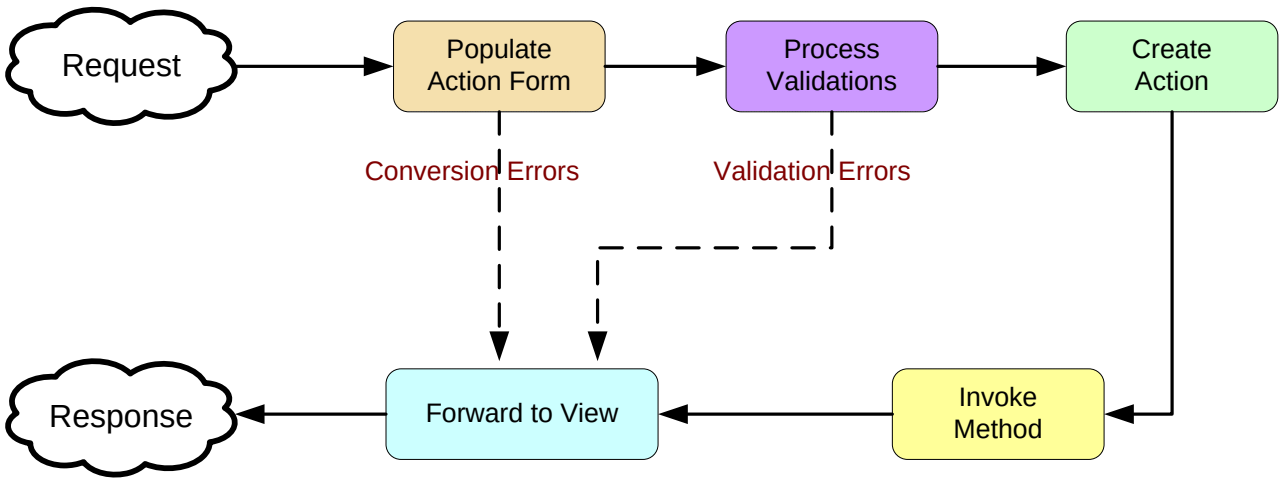


Figura 3: Proceso de petición en Struts

Utiliza el patrón Facade2 para implementar las acciones que deben ejecutarse bajo el modelo de negocio, basándose en clases predefinidas por el framework.

La generación de interfaces se realiza utilizando un conjunto de etiquetas predefinidas por Struts, con el objetivo de evitar el uso de scriptlets por temas de mantenimiento.

En la siguiente tabla podemos ver la correspondencia entre las clases de Struts y los componentes del patrón MVC

Clase	Descripción
ActionForward	Una acción de usuario o selección de una vista.
ActionForm	Los datos para un cambio de estado.
ActionMapping	El evento producido por un cambio de estado.
ActionServlet	Es la parte del Controlador que recibe las acciones de usuario, los cambios de estado y las selecciones de vistas.
Action classes	Es la parte del Controlador que interactúa con el modelo para realizar un cambio de estado o una consulta, y comunicar al ActionServlet cuál será la siguiente vista a seleccionar.

Tabla 2: Clases de Struts – componentes Patrón MVC

Además de las clases anteriores Struts utiliza una serie de archivos de configuración para unir el Controlador y el Modelo. En la siguiente tabla podemos ver estos archivos de configuración con su correspondiente rol.

Archivo	Rol
ApplicationResources.properties	Guarda los mensajes y etiquetas necesarios para que el sistema pueda internacionalizarse.
Struts-config.xml	Guarda la configuración por defecto de los objetos del controlador. Incluye: las acciones de usuario, cambios de estado, y consultas de estados soportadas por el modelo.

Tabla 3: Struts – archivos de configuración

Aspectos destacables

Algunos de los aspectos más destacables que convierten a Struts en uno de los frameworks más implementados en la actualidad son:

- *HTTP-centric*: Struts ha sido diseñado sobre el modelo HTTP request-response, que ya conocen la mayoría de los desrrolladores.
- *Logging Estándar*: Puede utilizar el contenedor por defecto del sistema, por lo que no necesita ningún componente adicional.
- *Debug logging opcional*: Adicionalmente puede crear logs de mensajes de estado que pueden ayudar al proceso de depuración.
- *Model neutral*: No depende de ninguna capa de persistencia en particular.

- *Configuración centralizada*: Struts encapsula los detalles de la implementación de una aplicación o un módulo, en una única configuración.
- *Diferentes mensajes de recursos para cada escenario*: Los traductores pueden trabajar independientemente en sus ficheros de mensajes de recursos. Añadir soporte para un nuevo escenario, es tan simple como añadir un nuevo fichero.
- *Ligero*: Su núcleo tiene relativamente pocas clases. Por lo que no es complicado de aprender.
- *Código abierto*: Con licencia Apache.
- *Comunidad de desarrolladores*: Existe una gran comunidad de desarrolladores detrás de Struts. Existen gran cantidad de extensiones.
- *Servicios*: Struts forma parte de gran cantidad de productos del mercado, como por ejemplo WebSphere de IBM. Además existen gran cantidad de utilidades de diversas empresas especializadas.
- *Equipo de desarrollo*: Struts dispone de su propio equipo de desarrollo.
- *Soporte*: Existe un foro administrado por profesionales. Además de una lista de correo y al menos dos portales web de soporte. Hay disponible una amplia bibliografía sobre Struts.
- *Versiones estables*: Antes de publicar una nueva versión, ésta ha pasado un período amplio de pruebas para asegurar un producto final de calidad.
- *Soporte i18n*: Soporte para la internacionalización de las aplicaciones.

- *Documentación*: La documentación disponible es extensa y detallada, lo que evita en la mayoría de los casos tener que consultar el código.
- *Patrones de diseño*: Implementa varios patrones de diseño clásicos, que son familiares a los desarrolladores.
- *Extensible*: Pueden configurarse todas las opciones por defecto. Un desarrollador puede personalizar clases como ActionForm y Action.

Puntos débiles

En gran parte los puntos débiles están estrechamente relacionados con los aspectos tratados anteriormente:

- *Eventos*: Struts está estrechamente ligado al modelo request-response utilizado por HTTP, por lo que no dispone de un modelo de eventos.
- *Debugging*: No dispone de un soporte automático para debugging. Los desarrolladores deben crear de forma manual los puntos de ruptura, escribiendo sobre la salida estándar o en el fichero de log.
- *Modelo de datos*: El acceso a la capa de persistencia es responsabilidad del desarrollador.
- *ActionServlet único*: Sólo puede utilizarse un único ActionServlet en una aplicación.
- *Conocimiento previo*: Para trabajar con Struts, es necesario conocer varias de sus clases especiales y cómo interactúan.
- *Soporte oficial*: Apache Software Foundation es una organización formada por voluntarios, no dispone de una plantilla fija que pueda ofrecer una garantía de respuesta.

- *Lista de correo desproporcionada*: Debido al gran número de usuarios de la lista de correo, se hace difícil localizar la información.
- *Versiones finales lentas*: La salida de versiones nuevas estables es más lenta de la habitual en este tipo de proyectos.
- *Limitaciones i18n*: Existen limitaciones para el manejo de grandes bloques de texto.
- *Pensado para JSP*: Aunque se utiliza el patrón MVC, existe una predisposición al uso de JSP.
- *Falta de traducción*: Muchos de los mensajes del sistema como las excepciones JSP sólo se muestran en inglés.

Spring

Definición

Spring es un framework de código abierto, creado por Rod Johnson con la idea de facilitar el desarrollo de aplicaciones empresariales. Y lo consigue gracias al uso de JavaBeans para hacer cosas que antes debían hacerse con EJB's.

Spring está formado por varios módulos. Si los utilizamos todos nos permite desarrollar aplicaciones empresariales, pero también tenemos la opción de utilizar sólo los módulos que sean necesarios para nuestra aplicación y utilizar otras opciones si consideramos que Spring no cubre las nuestras expectativas. Es por ello que Spring ofrece integración con otros muchos frameworks y librerías.

En la siguiente figura podemos observar la distribución de módulos de Spring.

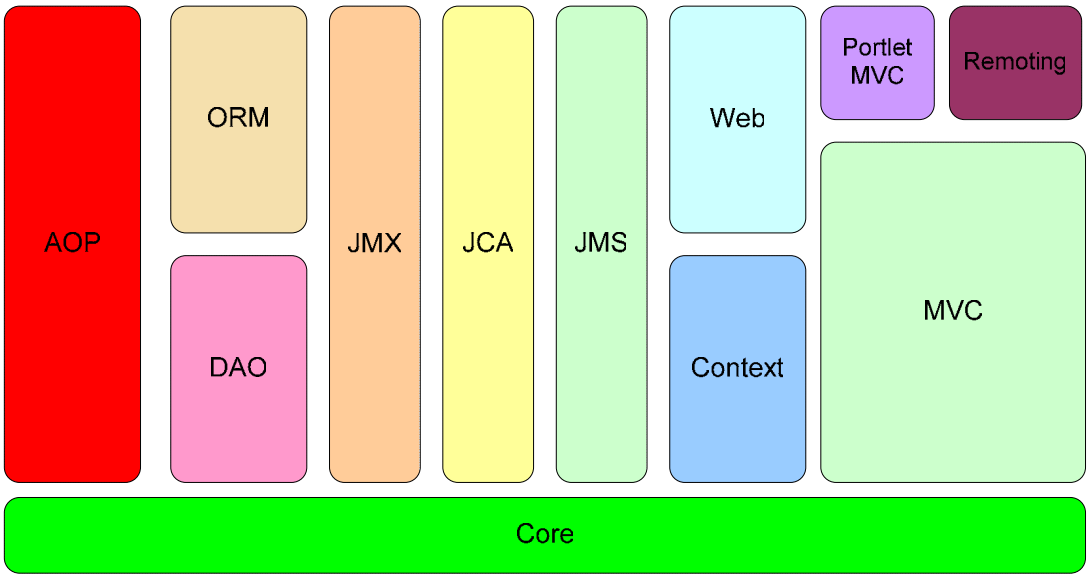


Figura 4:Módulos de Spring

Implementación del patrón MVC en Spring

Como ocurre en la mayoría de los frameworks que implementan MVC, Spring tiene un servlet que hace las funciones de Front Controller. En Spring este servlet recibe el nombre de Dispatcher Servlet y se encarga de controlar cada una de las peticiones (request) que realiza el usuario.

Utilizaremos la siguiente figura para ilustrar el flujo de una petición:

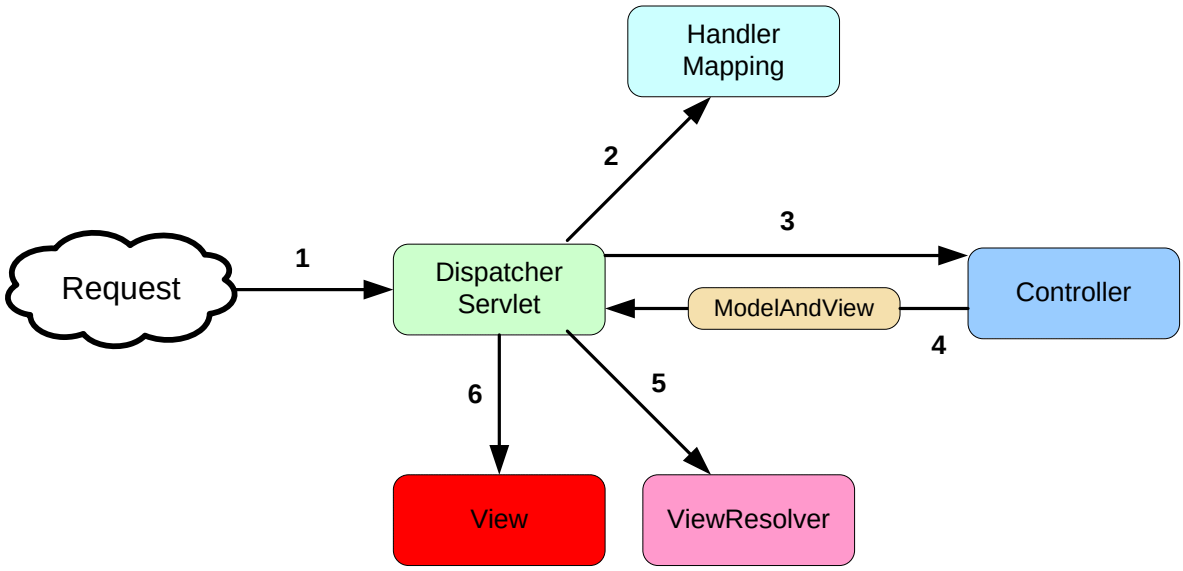


Figura 5:Flujo de petición en Spring

1) La petición llega al Dispatcher Servlet, que se encargará de delegar el procesamiento de la petición al componente correspondiente.

2) Para saber cuál será el nombre del componente que deberá recibir la petición, Spring utiliza el Handler Mapping, cuya función es la de determinar cuál será el Controller que recibirá la petición.

Existen varios Handler Mapping dependiendo de la capacidad que tengan para mapear controladores. En la siguiente tabla se muestran los distintos tipos de Handler Mapping:

Handler Mapping	Cómo mapea el Request
BeanNameUrlHandlerMapping	Mapea controladores a URL basándose en el nombre del Bean
SimpleUrlHandlerMapping	Mapea controladores a URL basándose en una colección de propiedades que se definen en el Spring application context.
ControllerClassNameHandlerMapping	Mapea controladores a URL utilizando el Controller Class Name
CommonsPathMapHandlerMapping	Mapea controladores a URL usando metadatas en el código del controlador. Los metadatas se definen utilizando Jakarta Commons Atributes.

Tabla 4: Spring – HandlerMappings

3) Una vez que el Handler Mapping ha proporcionado el nombre del controlador que se hará cargo de la petición, el Dispatcher se encarga de enviar al Controller correspondiente dicha petición.

La creación de un Controller en Spring requiere que se cree una clase que herede de los Controller existentes en Spring, por lo que deberá utilizarse el adecuado, dependiendo de su funcionalidad. En la siguiente tabla se muestran los distintos tipos de Controller definidos en Spring:

Controller type	Clase	Cuándo se usa
View	ParameterizableViewController UrlFilenameViewController	Cuando un controlador solo necesita desplegar información.
Simple	Controller (interface) AbstractController	Para controladores simples que solo se utilizan como Simples Servlet.
Throwaway	ThrowawayController	Para manejar los request como un comando.
Multiaction	MultiActionController	Para implementar una serie de acciones con similar lógica.
Command	BaseCommandController AbstractCommandController	Si los controladores reciben parámetros estos son manejados dentro de un objeto.
Form	AbstractFormController SimpleFormController	Para desplegar y procesar un formulario, bajo el mismo componente.

Tabla 5: Spring – Tipos de Controller

4) Cuando el Controller recibe la petición, se crea un objeto llamado ModelandView que se encarga de:

- Entregar un nombre lógico a la vista que deberá realizar el despliegue del Model
- Entregar un nombre lógico al Model asociado a este componente
- Inyectar el objeto Model que posee los datos que serán mostrados en la vista

5) De vuelta en el Dispatcher, éste debe delegar la responsabilidad del mapeo del nombre lógico de la vista al ViewResolver. Que se encargará de realizar el mapping entre el nombre lógico de la vista y el componente.

Spring dispone de una serie ViewResolver tal y como muestra la siguiente tabla:

View Resolver	Cómo trabaja
InternalResourceViewResolver	Resuelve el nombre lógico utilizando el mapping a velocity y JSP.
BeanNameViewResolver	Resuelve el nombre lógico utilizando Beans definidos en el Spring Context.
ResourceBundleViewResolver	Define el mapping entre los nombres lógicos y las vistas asociadas , definiéndolo en un archivo de

XmlViewResolver	propiedades. Define el mapping entre los nombres lógicos y las vistas asociadas , definiéndolo en un archivo XML.

Tabla 6: Spring – Tipos de ViewResolver

6) Cuando la Vista ha realizado su proceso, el Dispatcher envía la petición de vuelta al usuario.

Aspectos destacables

- *Implementación MVC*: Spring ofrece una división limpia entre Controllers, Models (JavaBeans) y Views.
- *Flexible*: Spring es muy flexible, ya que implementa toda su estructura mediante interfaces. Además, todas las partes del framework son configurables vía plug-in en la interface, aunque Spring provee clases concretas como opción de implementación.
- Spring MVC provee interceptores también como controllers que permiten factorizar el comportamiento común en el manejo de múltiples requests.
- *Integración*: Spring no obliga a utilizar JSP, permite utilizar XLST, Velocity o implementar un lenguaje propio para integrarlo en la View de la aplicación.
- *Fácil de testear*: Los controllers de Spring se configuran mediante IoC al igual que el resto de objetos, lo que los hace fácilmente testeables e integrables con otros objetos que estén en el contexto de Spring, y por tanto sean manejables por éste.

Las partes de Spring son fácilmente testeables, ya que se evita la herencia de una clase de manera forzosa y una dependencia directa en el controller del servlet que maneja las peticiones.

- *Buena arquitectura*: La capa Web de Spring es una pequeña parte situada encima de la capa de negocio, lo que parece un avance con respecto a otros frameworks web que dejan a tu elección la implementación de los objetos de negocio, mientras que Spring ofrece un framework para todas las capas de la aplicación.
- *Más cantidad de código testeable*: Las validaciones no dependen de la Api de servlets.
- *Interfaz bien definida*: Spring tiene una interfaz bien definida para la capa de negocio.

Puntos débiles

- Necesita gran cantidad de archivos XML para su configuración.
- No dispone de soporte nativo para AJAX.
- Es demasiado flexible, tanto que no existe un Controller padre.

JavaServer Faces

Definición

JavaServer Faces es un framework para el desarrollo de aplicaciones web creado por Sun Microsystems, basado en el patrón MVC (Modelo Vista Controlador).

Al igual que Struts, JSF pretende normalizar y estandarizar el desarrollo de aplicaciones web. Debemos tener en cuenta que JSF es posterior a Struts, y por lo tanto se ha nutrido de la experiencia de éste, mejorando algunas sus deficiencias. Es más, el creador de Struts (Craig R. McClanahan) también es líder de la especificación de JSF.

Es un framework orientado a la interfaz gráfica de usuario (GUI), que tiene como objetivo facilitar el desarrollo de las mismas. Implementa, tal y como hemos comentado el patrón de diseño MVC, realizando una separación entre comportamiento y presentación y proporcionando su propio servlet como controlador. Todo ello permite un desarrollo más sencillo y una estructuración del sistema más lógica.

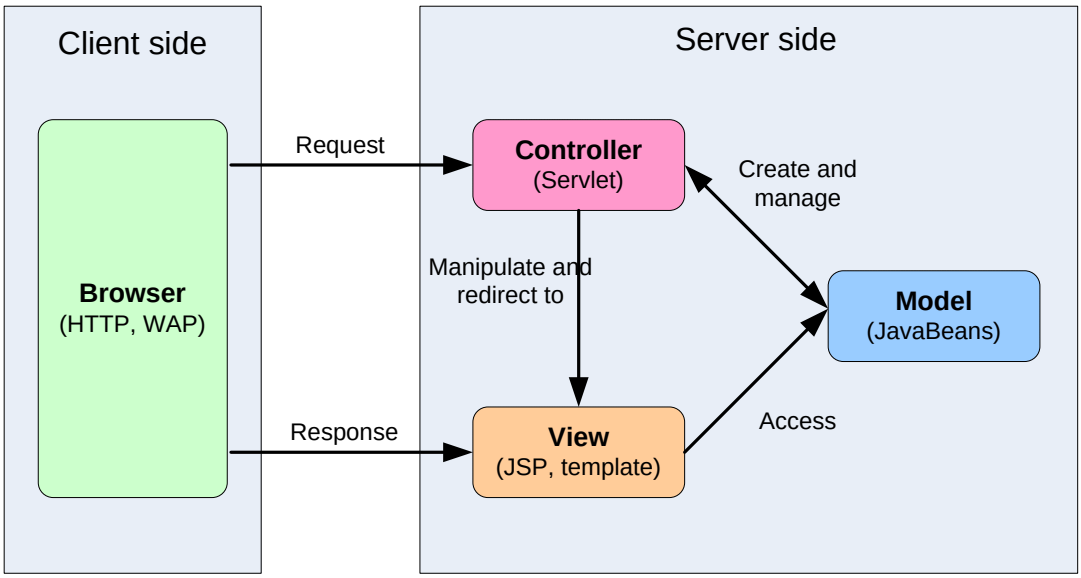


Figura 6:Variación de MVC implementada en JSF

Como puede observarse, el enfoque no es nuevo, lo que hace de JSF un framework atractivo es que ofrece un modelo basado en componentes para el desarrollo de aplicaciones web, que lo asemeja a las herramientas de desarrollo utilizadas hasta ahora para el desarrollo de aplicaciones GUI.

En JSF, los componentes GUI de la página están formados por objetos con su propio estado, por lo que podemos trabajar con los estados de los objetos y conectar los eventos producidos por la interacción del usuario con las acciones que les correspondan.

JSF define tres de las cuatro capas RAD (Rapid Application Development): Una arquitectura de componentes, un conjunto de UI widgets estándar y una infraestructura para la aplicación.

Funcionamiento de JSF

El funcionamiento del ciclo de vida es similar al de una página JSP, es decir, el cliente hace la petición http (request) y el servidor responde con un resultado (response). La diferencia está en que el ciclo de vida de JSF incluye nuevos pasos.

En el siguiente gráfico podemos observar cuál sería el proceso de una petición estándar. Existen seis fases y la ejecución de cualquier evento que pueda producirse durante el ciclo de vida.

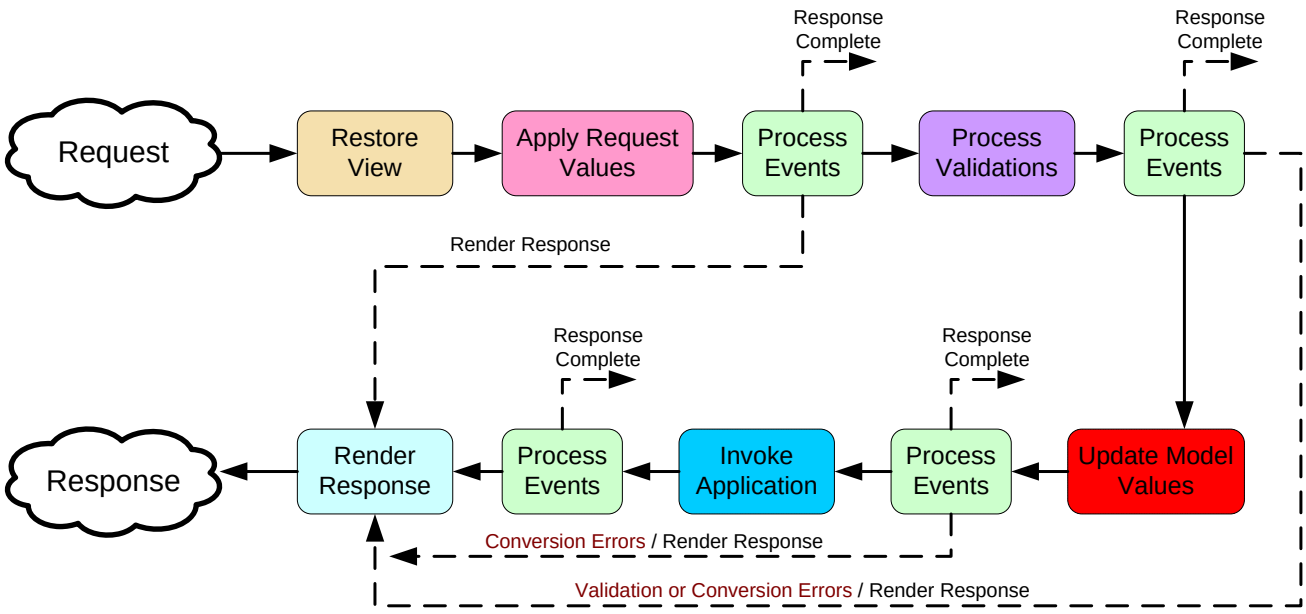


Figura 7: Proceso de petición en JSF

Restore View: Es la primera etapa que se realiza y comienza cuando se realiza una petición. Su cometido es el de crear un árbol que incluya todos los componentes de la página.

Apply Request Values: Se le asigna a cada uno de los componentes del árbol el valor que le corresponda tras la petición y se almacena.

Process Validations: Una vez almacenados los valores de los componentes, se validan según las reglas creadas para ello.

Update Model Vallues: Se utilizan los valores locales de los componentes para actualizar los beans que están relacionados con esos componentes. A esta etapa se llega sólo en el caso de que las validaciones anteriores se hayan podido realizar con éxito.

Invoke Application: Se ejecutan las acciones correspondientes al evento inicial que originó el proceso.

Render Response: Se renderiza la respuesta y se devuelve al cliente.

Aspectos destacables

- *Facilidad de uso*: JSF ofrece una clara separación entre la lógica y la presentación que tradicinalmente ofrece la arquitectura UI, permitiendo a cada miembro del equipo de desarrollo enfocarse en su parte del proceso de desarrollo. Proporciona además, un sencillo modelo de programación para enlazar todas las piezas.
- *Estandarización*: Es una tecnología desarrollada por la Java Community Process. Se encuentra definida por una serie de especificaciones: JSR 127, JSR 252, JSR 276.

- *Independencia del dispositivo*: Es un framework flexible, permite al desarrollador crear sus propios componentes.

- *Nuevo concepto de Vista*: JSF trata la interfaz de usuario (Vista) de manera diferente a como lo hacen el resto de frameworks de aplicaciones web. Se asemeja más al estilo de la programación de GUI's, donde las interfaz se hace a través de componentes y está basada en eventos.

  Proporciona una rica arquitectura para manejar el estado de los componentes, procesar datos, validar la entrada del usuario, y manejar eventos.

- *Mejora los conceptos de componente-UI y capa web*: Sin limitar el uso de una tecnología de script en particular o un lenguaje de marcas determinado.

  Aunque incluye una librería de etiquetas JSP personalizadas, los APIs de JSF están creados directamente sobre el API JavaServlet, lo que permite, por ejemplo, utilizar otra tecnología de representación junto a JSP o generar salidas para diferentes dispositivos cliente.

Puntos débiles

- JSF no puede competir en madurez con el resto de los frameworks, pero si puede ser una opción muy recomendable para nuevos desarrollos.

- Aunque tal y como hemos comentado es un framework sencillo de utilizar, requiere una curva de aprendizaje algo superior al resto, debido en gran medida a que no es todo lo intuitivo que puede parecer.
- Hace un uso intensivo de JavaScript.
- No maneja URL's, por lo que no existe la opción de volver a la página anterior.
- Utilizar AJAX no es del todo sencillo, sobre todo en las primeras versiones.
- Al ser un framework de componentes, su éxito se basa en el número de componentes. Y este puede ser un problema, la falta de componentes disponibles.

Tecnología Adobe Flash

Historia

La tecnología Flash tiene sus orígenes en una pequeña compañía llamada FutureSplash, que fue adquirida por Macromedia en 1997 para enfocar a la web su programa de creaciones multimedia, Director.

Ya en el año 2005 Adobe Systems adquiere los productos de Macromedia, y pasa a denominarse Adobe Flash, que es el nombre que tiene en la actualidad.

Introducción

Adobe Flash es una aplicación que trabaja sobre fotogramas, destinada a la producción de animaciones. Utiliza para ello gráficos vectoriales e imágenes ráster, sonido, audio bidireccional, flujo de vídeo y su propio lenguaje script de programación: Action Script.

La reproducción de las animaciones se hacen utilizando Flash Placer, que no es más que una máquina virtual capaz de ejecutar los archivos generados con el entorno de desarrollo Flash.

La API de Adobe Flash está basada en JavaScript-C, por lo que los comandos C++ no se ejecutan directamente desde C/C++, sino a través de JavaScript. Esto da mayor flexibilidad al desarrollador al tratarse de código abierto interpretable.

ActionScript

La versión actual del lenguaje de programación de Flash, Action Script 3.0, es un lenguaje orientado a objetos que permite un mayor control, rehusabilidad de código y demás características de este tipo de lenguajes, respecto a las versiones anteriores.

Gracias a la tecnología Asynchronous Flash y XML, Flash puede utilizar sus librerias con capacidades XML para mostrar contenidos en el explorador. Esto permite añadir protección a los contenidos que se reproducen desde flash. Es la tecnología utilizada por ejemplo en youtube.com, que vemos cuando utilizamos el servicio, la información de vídeo y mp3 se envía parseada o mediante streams, de forma que es difícil de almacenar en el ordenador del cliente.

Formato

El uso de gráficos vectoriales unido al código de sus scripts permiten que el tamaño de sus dicheros sea reducido y que sus streams utilicen un ancho de banda inferior que si utilizáramos bitmaps o vídeo clips.

Además, el reproductor Flash Player incluye en sus últimas versiones una máquina virtual ActionScript (AVM: ActionScript Virtual Machina) que le permite interactividad en tiempo de ejecución, ofrece soporte para vídeo, mp3 y gráficos bitmap. Desde la versión 8 se ofrecen dos video codecs: On2

Technologies VP6 y Sorenson Spark, y soporte en tiempo de ejecución para JPEG, Progressive JPEG, PNG y GIF.

Comparativa de los frameworks

Con el fin de realizar una evaluación/comparación de los frameworks estudiados previamente, lo más exhaustiva e imparcial posible, y siempre desde el punto de vista del desarrollador, vamos a utilizar los siguientes criterios:

- Facilidad para utilizar listas de datos ordenadas y/o paginadas:

  Struts y Spring disponen de librerías de etiquetas (tag's) que les permiten integrar listas de datos con la funcionalidad de ordenar y controlar las páginas, por su parte, JavaServer Faces dispone de un control propio: dataTable sin posibilidad de ordenar, requiere que el desarrollador escriba su propa lógica si fuera necesario.

- Marcación de páginas para poder regresar a ellas (bookmarkability):

  Struts y Spring permiten un control total de URL's, por el contrario JavaServer Faces utiliza siempre el POST.

- Sencillez para realizar validación JavaScript en el cliente:

  El sistema de validación de Struts y Spring es un sistema maduro, ya que utilizan Commons Validator, mientras que JavaServer Faces incorpora mecanismos básicos para facilitar la validación de formularios que proporcionan servicio al lado del servidor, aunque puede utilizar también Commons Validator.

- Facilidad para testear:

  En cuanto a los sistemas de Test, Struts puede utilizar StrutsTestCase y Spring lo hace de forma sencilla con mocks (Spring Mocks). JavaServer Faces puede testear de forma sencilla.

- ¿Soporta el framework la integración con Spring? ¿De manera sencilla?:

  Podemos integrar Struts y JavaServer Faces con el framework Spring. Struts dispone de: ContextLoaderPlugin y las Clases de Support. Por su parte JavaServer Faces tiene: DelegatingVariableResolver o JSF-Spring Library.

- Soporte para la internacionalización de la aplicación:

  Los tres frameworks dan una solución óptima a la internacionalización, utilizando para ello un fichero por cada localización deseada.

- Diseño de las páginas:

  Los tres frameworks puede utilizar la librería de etiquetas Tiles para realizar un mejor diseño de la capa de presentación. Igualmente todos los frameworks pueden utilizar la tecnología siteMesh para mejorar el diseño de las páginas, necesitarán un posterior mantenimiento después de su puesta en funcionamiento.

- Herramientas disponibles que complementen el framework, en especial IDE's:

  Strurts dispone de gran cantidad de herramientas IDE, incluso existen frameworks desarrollados sobre ellas.

  Spring dispone de Spring IDE y JavaServer Faces tiene muchas herramientas y cada vez más y mejores.

- Demanda del uso del framework en el mercado laboral:

  Hemos consultado el número de ofertas de trabajo publicadas en infoojbs.net en el mes de abril para confeccionar el siguiente gráfico:

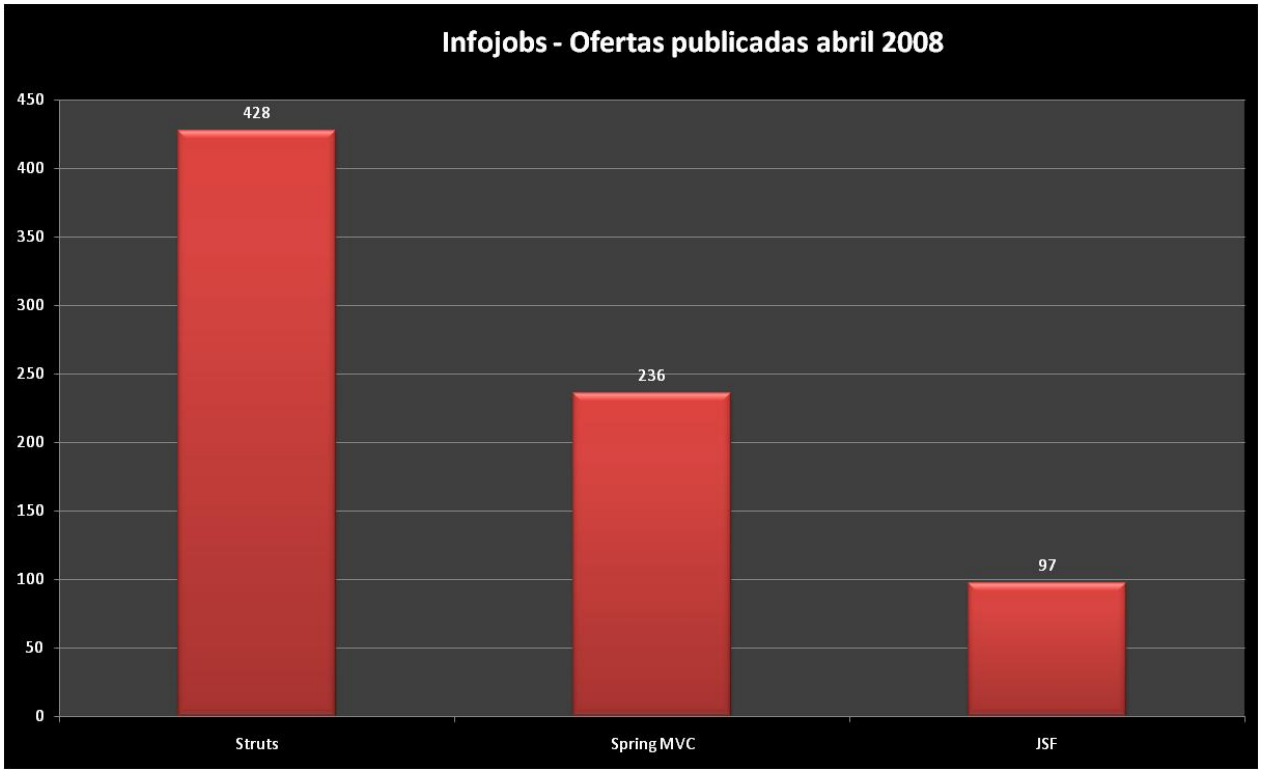


Figura 8: Ofertas de trabajo publicadas abril 2008

Como se puede observar Struts es el framework más demandado, quizás debido a que sea el que mayor número de aplicaciones tiene

instaladas en el mercado nacional, debido a que es el más antiguo y maduro de los frameworks.

- Documentación:

  Para hacer esta comparativa hemos consultado el número de referencias de libros a la venta en amazon.com. Como puede observarse en el gráfico situado más abajo, es Struts el que mayor bibliografía tiene disponible, seguido de Spring y JavaServer Faces.

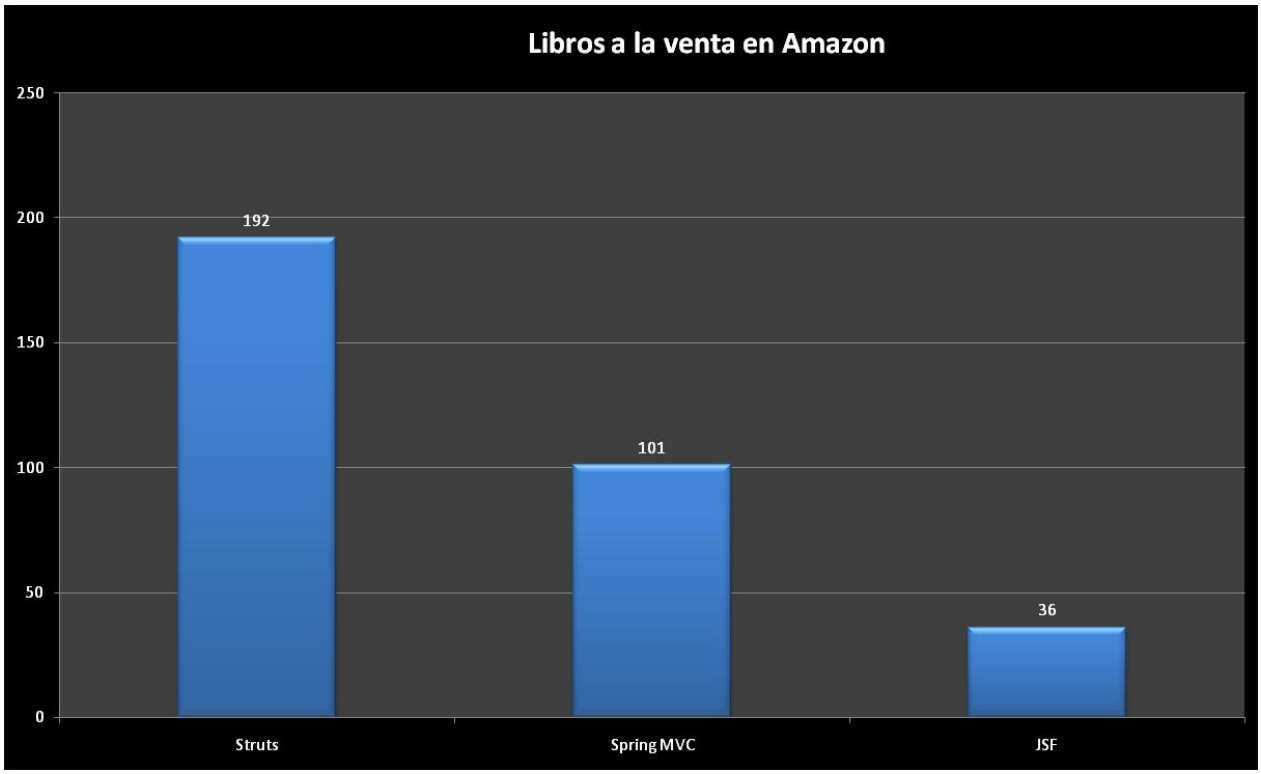


Figura 9: Libros a la venta en amazon.com

Capítulo 4: Análisis y Diseño del Componente

Análisis del componente

Introducción

A la vista del análisis sobre los frameworks realizado anteriormente, hemos elegido JavaServer Faces para implementar el componente, por ser un framework orientado a la interfaz de usuario.

El componente permitirá al Framework JSF realizar una representación gráfica, con salida en formato Adobe Flash, de un conjunto de datos suministrado, según la configuración indicada.

Para realizar la salida en el formato especificado utilizaremos el componente opensource Open Flash Chart.

Funcionalidades

La funcionalidad del componente será la de permitir la representación gráfica de un conjunto de datos en diferentes formatos.

El componente nos permitirá realizar las siguientes acciones:

- Visualizar los datos en distintos formatos gráficos: Líneas, Columnas, Columnas 3D, Circular, Áreas, Dispersión y Cotizaciones.
- Seleccionar el subtipo de gráfico deseado entre los siguientes:
  - Subtipos para el tipo Línea: Normal, Puntos, Puntos abiertos.
  - Subtipos para el tipo Columnas: Normal, Cristal, Degradado, Boceto.
- Especificar un título para el gráfico
- Seleccionar los colores de textos, ejes y fondos.
- Indicar el color y transparencia para las series de datos.
- Definir nombres de las categorías de datos a representar

- Especificar el tamaño (altura y anchura) de la película (swf) que contendrá la representación gráfica.
- Seleccionar una imagen o un degradado para el fondo de la gráfica.
- Personalizar la apariencia de los ejes: Número de divisiones, etiquetas, formatos numéricos.
- Especificar enlaces web para cada dato representado en los tipos: Líneas, Barras y Circular.

Actores

Desarrollador: Es el encargado de proporcionar los parámetros de configuración necesarios para que el componente en su ejecución, pueda proporcionar la representación gráfica de los datos suministrados.

Casos de uso

Descripción de los casos de uso

Caso 1: Representación gráfica de los datos	
Funcionalidad:	Describe la funcionalidad de representar gráficamente en formato Flash los datos suministrados.
Actores:	Desarrollador
Casos relacionados:	-
Precondición:	Requiere que previamente se suministren los datos de configuración necesarios, referidos a las distintas opciones que permite el componente: ▪ Tipo y subtipo de formato gráfico. ▪ Datos a representar. ▪ Título para el gráfico. ▪ Color para los textos.

	▪ Color/es para el fondo. ▪ Color de la serie de datos. ▪ Imagen para el fondo. ▪ Especificar los nombres de las categorías de datos a representar. ▪ Formato de los ejes. ▪ Tamaño de la película (swf) que contendrá la representación gráfica.
Poscondición:	Como resultado se obtendrá la representación gráfica de los datos suministrados de acorde a la configuración especificada.
Descripción:	El desarrollador deberá configurar los parámetros solicitados para que el componente sea capaz de generar la gráfica solicitada.

Diagramas de los casos de uso

Caso 1: Representación gráfica de los datos

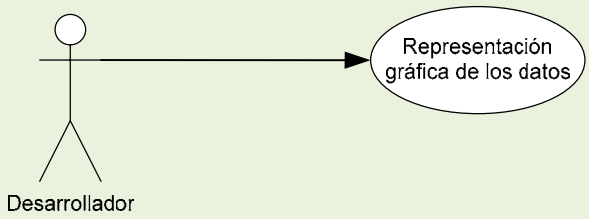


Figura 10: Caso de uso Representación gráfica de los datos

Diseño del componente

Diagrama de clases

En la figura 11 podemos ver el diagrama de clases del componente, en el que observamos las distintas clases utilizadas para realizar la función descrita.

Debemos partir de la idea de que en JavaServer Faces, todos los componentes UI deben extender de la clase abstracta `javax.faces.component.UIComponent`, por lo que la clase principal del diseño de nuestro componente, UIFlashChart, extiende de UIComponentBase que extiende a su vez de UIComponent.

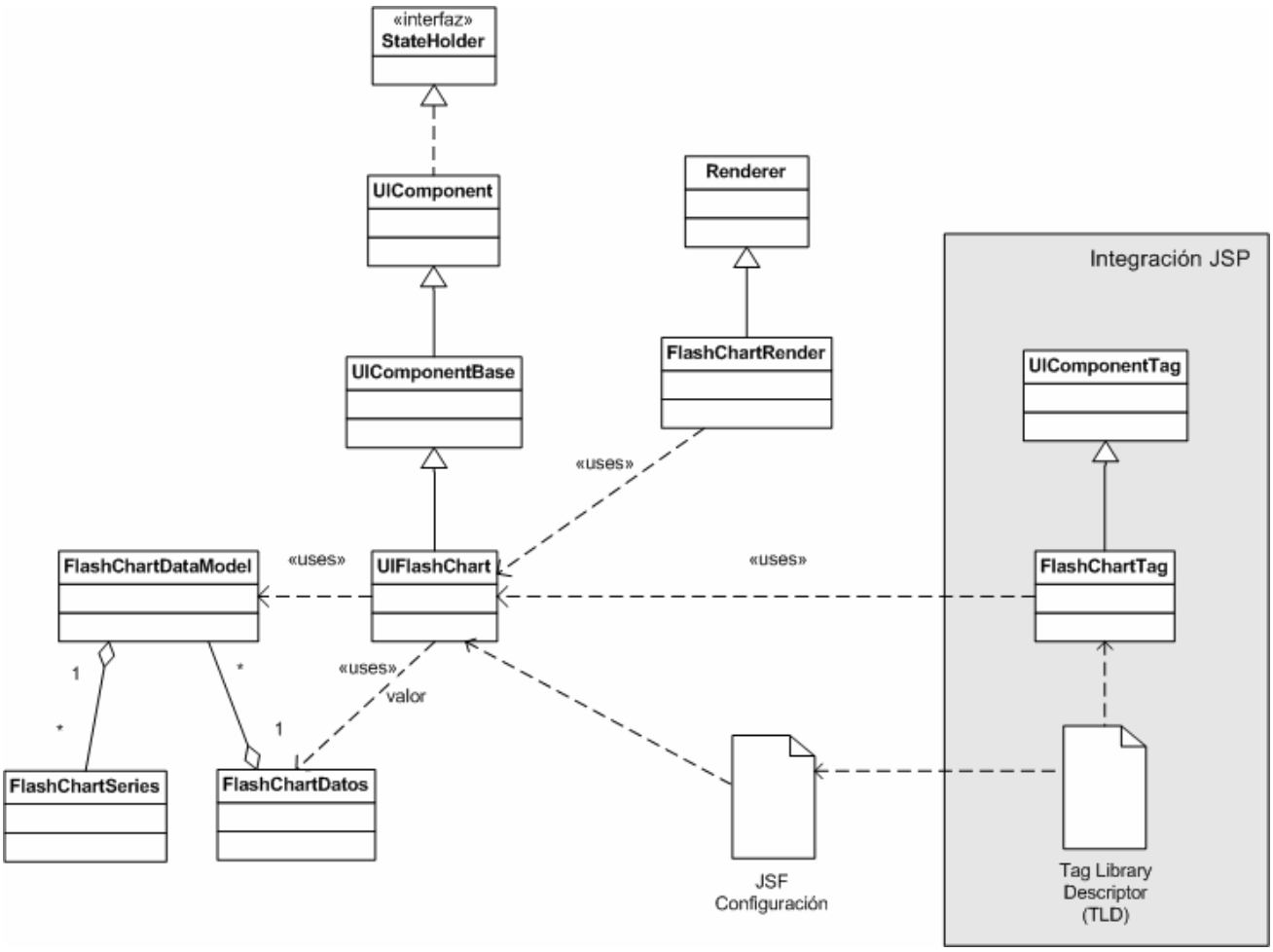


Figura 11:Diagrama estático del análisis

El componente debe encargarse de representar gráficamente en formato flash, las distintas series de datos que se le suministren, ello implica que en una misma gráfica puedan representarse varias series de datos independientes y en distintos formatos, por ejemplo, una serie de datos en formato barras y otra en formato líneas.

Por lo que cada serie de datos mostrada a través de UIFlashChart, la representamos en nuestro diseño, mediante la clase FlashChartDataModel.

Un FlashChartDataModel contendrá la referencia a los datos, los enlaces y las propiedades de la serie a representar, por lo que dispone de las propiedades que muestra la tabla 7.

Propiedad	Tipo	Descripción
id	Integer	Identificador de la serie de datos
datos	List	Conjunto de datos a representar
links	List	Contiene una lista de enlaces para cada uno de los datos a representar.
series	FlasChartSeries	Referencia a las propiedades de la serie que se desea representar.

Tabla 7:Propiedades de la cases FlashChartDataModel

Cada una de las referencias de la propiedad series viene representada por la clase FlashChartSeries. De esta forma, podremos especificar cada una de las distintas posibilidades de las que dispone el componente para definir una serie de datos.

Nuestro componente será capaz de representar hasta 12 tipos de series distintas, dependiendo del tipo de gráfica que deseamos formar. Cada tipo de serie requerirá especificar entre tres y seis parámetros distintos. En la tabla 8, se muestran los distintos tipos de series y sus posibles parámetros de configuración.

Tipo	Parámetros	Descripción
line	Ancho, color, leyenda [,tamaño]	Líneas
line_dot	Ancho, color, leyenda, tamaño punto	Líneas con puntos rellenos
line_hollow	Ancho, color, leyenda, tamaño punto	Líneas con puntos huecos
bar	Alpha, color, leyenda, tamaño texto	Columnas
bar_glass	Alpha, color, color, leyenda, tam. Txt	Columnas efecto cristal
bar_fade	Alpha, color, leyenda, tamaño texto	Columnas efecto degradado
bar_sketck	Alpha, color, leyenda, tamaño texto	Columnas efecto boceto
bar_3d	Alpha, color, leyenda, tamaño texto	Columnas efecto tres dimensiones
area_hollow	Tamaño, tam. punto, alpha, color [,leyenda] [,tam. txt]	Áreas
pie	Alpha, color, estilo	Circular
scatter	Ancho, color, leyenda [,tamaño]	Dispersión
hlc	Alpha, ancho, color [,leyenda] [,tamaño]	Cotizaciones

Tabla 8:Tipos de series

Atendiendo a esto, se han definido ocho propiedades en la clase FlashChartSeries y distintos constructores para especificar los tipos de series de datos diferentes. La tabla 9, muestra las propiedades de la clase.

Propiedad	Tipo	Descripción
id	Integer	Identificador de la serie
tipo	String	Especifica el tipo de serie a representar
p1	String	Contiene el valor del primer parámetro de la serie.
p2	String	Contiene el valor del segundo parámetro de la serie.
p3	String	Contiene el valor del tercer parámetro de la serie.
p4	String	Contiene el valor del cuarto parámetro de la serie.
p5	String	Contiene el valor del quinto parámetro de la serie.
p6	String	Contiene el valor del sexto parámetro de la serie.

Tabla 9:Propiedades de la cases FlashChartSeries

Para poder representar varias series de datos en un mismo gráfico, se define a clase FlashChartDatos, que contiene las distintas instancias de la clase FlashChartDataModel que se hayan definido. Esta clase es una subclase de `java.util.ArrayList`.

Se ha definido un render específico para el componente, para ello se ha definido la clase FlashCharRenderer, que extiende de la clase Renderer,

para que sea la encargada de generar la salida deseada, que en nuestro caso es una etiqueta <object> html.

En concreto deberá generar una etiqueta con la siguiente estructura:

```
<object id="" classid="clsid:D27CDB6E-AE6D-11cf-96B8-
444553540000" width="" height="" style="undefined">

<param name="movie" value="open-flash-chart.swf" />
   <param name="bgcolor" value="" />
   <param name="quality" value="" />
   <param name="flashvars" value="variables=" />
</object>
```

Por último, la clase FlashChartTag nos permitirá registrar el componente y su render.

Implementación

En cuanto a la implementación de nuestro componente, pasamos a destacar algunos de los aspectos más importantes de la misma.

Como el componente no es una subclase de ninguna clase de componente estándar, debemos antes de nada, definir tanto la familia como el tipo de componente que queremos definir, para ello definimos:

```
public static final String COMPONENT_FAMILY = "flash.Chart";
public static final String COMPONENT_TYPE = "flash.Chart";

public String getFamily(){
   return "flash.Chart";
}
```

En el constructor, indicamos cuál será el renderer por defecto que utilizará el componente:

```
    public static final String COMPONENT_FAMILY = "flash.Chart";
    public static final String COMPONENT_TYPE = "flash.Chart";

    public UIFlashChart() {
        super();
        setRendererType(DEFAULT_RENDERER);
    }
```

La implementación de los métodos de control del Estado se hacen de forma estándar, creamos los métodos saveState y restoreState.

En el método saveState guardamos el estado utilizando un array de objetos, asignando las propiedades a cada una de las posiciones del array, empezando por el estado de la superclase. Y en el método restoreState sacamos los valores del array.

El registro del componente requiere que declaremos un componente de tipo flash.Chart tal y como habíamos declarado en la constante *COMPONENT_TYPE*.

```
    <component>
        <component-type>flash.Chart</component-type>
        <component-class> uoc.jgarciadez.flashchart.component.UIFlashChart
        </component-class>
    </component>
```

La clase FlashChartRenderer extiende directamente de la clase abstracta Renderer. Como UIFlashChart se encarga de generar un etiqueta <object>, se han implementado únicamente los métodos encodeBegin y encodeEnd.

Dentro de encodeBegin, se genera la cabecera y los parámetros de configuración que permitirán la representación gráfica y en encodeEnd se cierra la etiqueta object.

```
public void encodeBegin(FacesContext context, UIComponent component)
throws IOException {
    ResponseWriter writer = context.getResponseWriter();
    UIFlashChart chart = (UIFlashChart) component;
    writeCabecera(context, chart);
    writeParametros(context, chart);
}

public void encodeEnd(FacesContext context, UIComponent component)
throws IOException {
    ResponseWriter writer = context.getResponseWriter();
    writer.endElement("object");
}
```

Para registrar el renderer, utilizamos el mismo tipo que definimos en el constructor de UIFlashChart: flash.Chart.

```
<render-kit>
    <renderer>
            <component-family>flash.Chart</component-family>
            <renderer-type>flash.Chart</renderer-type>
            <renderer-class>
                    uoc.jgarciadez.flashchart.render.FlashChartRenderer
            </renderer-class>
    </renderer>
</render-kit>
```

Para conseguir la integración del componente con las páginas JSP, hemos tenido que definir la clase FlashChartTag que extiende directamente de UIComponentTag.

Se ha diseñado una etiqueta específica para nuestro componente y por tanto se ha definido un TLD (Tag Library Descriptor) para la misma, en el que aparecen definidos todos y cada uno de los atributos que mediante la etiqueta podrán definirse en nuestro componente.

A continuación se muestran algunas partes de la definición del TLD:

```
<taglib>
   <tlib-version>0.03</tlib-version>
   <jsp-version>1.2</jsp-version>
   <short-name>jgarciadez</short-name>
   <uri>uoc-flash-chart-component</uri>
   <description>jgarciadez tags</description>

...
```

```
<!-- Definición de la etiqueta en el tld -->
<tag>
      <name>fchart</name>
      <tag-class>uoc.jgarciadez.flashchart.tag.FlashChartTag</tag-class>
     <!-- Añadimos los atributos a la etiqueta fchart -->
      <attribute>
         <name>nombre</name>
         <description>Permite distinguir una gráfica de
otra</description>
      </attribute>
      <attribute>
         <name>datos</name>
         <description>Datos a representar</description>
      </attribute>
      <attribute>
         <name>tipo</name>
         <description>Tipo de gráfica</description>
      </attribute>
```

```
        <attribute>
           <name>ancho</name>
           <description>Especifique el ancho de la gráfica</description>
        </attribute>
        <attribute>
           <name>alto</name>
           <description>Especifique el alto de la gráfica</description>
        </attribute>
        <attribute>
           <name>titulo</name>
           <description>Especifique el titulo</description>
        </attribute>
        <attribute>
           <name>xleyenda</name>
           <description>Rótulo del eje X</description>
        </attribute>
        <attribute>
           <name>xetiquetas</name>
           <description>Etiquetas del eje X</description>
        </attribute>

   ...

      </tag>
   </taglib>
```

Esta definición nos permitirá utilizar nuestro componente con la etiqueta siguiente, por ejemplo:

```
   <jgarciadez:fchart datos="#{items}" titulo="Título de ejemplo,
   {font-size: 20px; color: #736AFF}" ancho="500" alto="500"
   xleyenda="Rótulo eje X, 32,  #736AFF, ndad" xcolor=""/>
```

Como hemos diseñado la etiqueta específicamente para el tipo de componente UIFlashChart y el tipo de renderer FlashChartRenderer, lo primero que debemos hacer es establecer estas propiedades:

```
    public String getComponentType() {
        return "flash.Chart";
    }

    public String getRendererType() {
        return UIFlashChart.DEFAULT_RENDERER;
    }
```

Una vez definidos los tipos de objetos que soportará, debemos establecer el método setProperties. Cabe mencionar que para poder establecer el valor de los datos, utilizamos su value-binding.

```
    if (datos != null){
        if (isValueReference(datos)){
                chart.setValueBinding("datos", app.createValueBinding(datos));
        }
        else
                throw new IllegalArgumentException("Error");
    }
```

Por último destacar que nuestro renderer debe tener en cuenta el browser (navegador) que utilice el cliente, ya que el resultado a generar será distinto, debido a que el objeto flash es diferente en Internet Explorer y los navegadores mozilla 5.0 (FireFox).

En el siguiente cuadro podemos ver el formato de objeto que debe generarse para que Internet Explorer muestre una película Flash.

```
<object id="" classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
width="" height="" style="undefined">
   <param name="movie" value="open-flash-chart.swf"/>
   <param name="bgcolor" value="#FFFFFF"/>
   <param name="quality" value="high"/>
   <param name="flashvars" value="variables=true&…"/>
</object>
```

El siguiente es el formato necesario para que sean los navegadores Mozilla 5.0 los que representen la película Flash:

```
<embed type="application/x-shockwave-flash" src="open-flash-chart.swf"
width="" height="" style="undefined" id=" " name=" " bgcolor="#FFFFFF"
quality="high" flashvars="variables=true&…">
</embed>
```

Para conocer cuál es el navegador utilizado por el cliente hemos implementado la siguiente función, que recoge los parámetros de la cabecera:

```
public static boolean isInternetExplorer(){
   String ua = getUserAgent();
   return (ua != null && ua.contains("MSIE"));
}


private static String getUserAgent(){
   FacesContext facesContext = FacesContext.getCurrentInstance();
   String ua = (String)facesContext.getExternalContext()
               .getRequestHeaderMap().get("User-Agent");
   return ua;
}
```

Capítulo 5: Análisis y Diseño de la aplicación ejemplo: GolfStat

Análisis de la aplicación ejemplo: GolfStat

Introducción

Con el fin de poder mostrar el uso del componente de la forma más exhaustiva posible, se ha pensado en desarrollar una pequeña aplicación que permita a los jugadores de golf, a partir de los datos que se generan en sus partidas, realizar un análisis gráfico de los resultados, para que les permita conocer mejor su juego y por tanto puedan mejorarlo.

Como el objetivo primordial de la aplicación es el de utilizar el componente diseñado, las partes correspondientes al mantenimiento de los datos no se implementarán completamente.

Funcionalidades

La aplicación se divide en tres subsistemas:

- Subsistema de Conexión: Encargado de controlar el acceso y la validación de los usuarios en el sistema. Su principal función es la de gestionar la seguridad de la aplicación.

- Subsistema de Mantenimiento: Será el encargado de realizar todas las acciones relacionadas con el mantenimiento de las entidades de datos.

- Subsistema de Servicios: Es la parte principal de nuestro diseño, ya que será la encargada de mostrar las gráficas a los usuarios, por lo que será la que hará uso de nuestro componente.

Pasemos a enumerar cada una de las funcionalidades de los subsistemas anteriores:

Subsistema de Conexión

- Autentificación de los usuarios de la aplicación: Se encargará de permitir la entrada a la aplicación, mediante el sistema de solicitud y verificación de nombre de usuario y contraseña.

Subsistema de Mantenimiento

- Alta de usuario: Esta opción permitirá añadir nuevos usuarios al sistema. Serán los propios usuarios los encargados de registrarse en el sistema.

- Alta de recorrido: Cada usuario podrá ir añadiendo cada uno de los recorridos que realice en las distintas partidas que juegue. Será obligatorio que indique los datos correspondientes al campo, tee de salida y fecha del recorrido.

- Baja de recorrido: Desde esta opción se podrá eliminar un recorrido. El usuario deberá indicar el código del recorrido que desea eliminar.

- Modificación de recorrido: Encargada de permitir al usuario la modificación de cualquiera de sus recorridos. Deberá indicarse el código del recorrido que desea modificarse.

- Consulta de recorrido: Esta opción permitirá ver los datos de un recorrido. El usuario deberá seleccionar el recorrido a mostrar de un listado.

Subsistema de Servicios

- Ver estadísticas: Esta opción mostrará al usuario su evolución en el juego. Mostrando diversas gráficas: comparación del recorrido con el par del campo, porcentajes de puntuación, comparación de medias de golpes, etc.

Actores

Usuario: Es el único actor del sistema, se encargará de realizar su registro en el sistema. Una vez registrado, podrá acceder a él para añadir nuevos recorridos, modificarlos, consultarlos o eliminarlos. Y podrá consultar las resultados estadísticos que le ofrece la aplicación a través de las representaciones gráficas.

Casos de uso

Descripción de los casos de uso

Caso 1: Login	
Funcionalidad:	Permite acceder a los usuarios a la aplicación.
Actores:	Usuario
Casos relacionados:	Alta de Usuario. Será necesario pasar por este caso de uso para poder ir al resto.
Precondición:	El usuario necesita haberse dado de alta previamente.
Poscondición:	Una vez introducidos el nombre usuario y la contraseña se accederá a la aplicación.
Descripción:	El usuario debe introducir su nombre de usuario y su contraseña. El sistema validará los datos introducidos y si estos son correctos le permitirá el acceso la aplicación. En caso contrario le denegará el acceso.

Caso 2: Alta de Usuario	
Funcionalidad:	Permite añadir usuarios al sistema.
Actores:	Usuario
Casos relacionados:	-
Precondición:	El usuario que se desea crear será identificado por un ID (nombre de usuario), éste debe ser válido y único en el sistema.
Poscondición:	En caso de que se hayan introducido todos los datos de forma correcta, se añadirá el nuevo

	usuario al sistema, en caso contrario de informará al usuario del error cometido.
Descripción:	El usuario deberá rellenar un formulario con los datos solicitados y pulsar sobre el botón “Guardar” para que se cree el nuevo usuario en el sistema. Si se pulsa el botón “Cancelar” se regresará a la ventana principal del sistema.

Caso 3: Alta de Recorrido	
Funcionalidad:	Permite que un usuario pueda añadir resultados de sus partidas al sistema.
Actores:	Usuario
Casos relacionados:	Login
Precondición:	El usuario ha debido autentificarse en el sistema. El nuevo recorrido será identificado por su ID, éste debe ser válido y único en el sistema.
Poscondición:	En caso de que se hayan introducido todos los datos de forma correcta, se añadirá el nuevo usuario al sistema, en caso contrario de informará al usuario del error cometido.
Descripción:	El usuario deberá rellenar un formulario con los datos solicitados y pulsar sobre el botón “Guardar” para que se cree el nuevo recorridoen el sistema. Si se pulsa el botón “Cancelar” se regresará a la ventana principal del sistema.

Caso 4: Baja de Recorrido	
Funcionalidad:	Permite eliminar un recorrido del sistema
Actores:	Usuario
Casos relacionados:	Login
Precondición:	El usuario ha debido autentificarse en el sistema. El recorrido que se desea eliminar de la base de datos será identificado por su ID, por lo que éste debe existir en el sistema.
Poscondición:	El recorrido es eliminado del sistema.
Descripción:	Se indicará el recorrido a eliminar, el sistema nos pedirá la confirmación de la eliminación del recorrido. En caso de que se acepte, el recorrido quedará eliminado, en caso contrario no se hará.

Caso 5: Modificación de Recorrido	
Funcionalidad:	Este caso de uso permite modificar todos los datos de un recorrido con excepción de su Identificador.
Actores:	Usuario
Casos relacionados:	Login
Precondición:	El usuario ha debido autentificarse en el sistema. El recorrido que se desea mnodificar será identificado por su ID, por lo que éste debe existir en el sistema.
Poscondición:	Se guardarán en el sistemas los nuevos datos del recorrido.
Descripción:	Se indicará el recorrido a modificar, aparecerán los

	datos actuales de dicho recorrido, el usuario podrá modificarlos. Para confirmar los cambios se pulsará el botón “Guardar”. Para volver al inicio sin modificar los cambios se pulsará el botón “Cancelar”.

Caso 6: Consulta de Recorrido	
Funcionalidad:	Permite visualizar todos los datos relativos a un recorrido.
Actores:	Usuario
Casos relacionados:	Login
Precondición:	El usuario ha debido autentificarse en el sistema. El recorrido a mostrar se identifica por su ID, por lo que éste debe existir en el sistema.
Poscondición:	Se muestran todos los datos del recorrido.
Descripción:	Se indicará el recorrido a mostrar, aparecerán los datos actuales de dicho recorrido.

Caso 7: Ver Estadísticas	
Funcionalidad:	Permitirá mostrar las representaciones gráficas de los datos de los recorridos de los usuarios.
Actores:	Usuario
Casos relacionados:	Login
Precondición:	Deberán existir datos a representar.
Poscondición:	Se muestra los datos de forma gráfica.

Descripción:	Se mostrarán al usuario las distintas gráficas de sus resultados.

Diagramas de los casos de uso

Casos de Uso

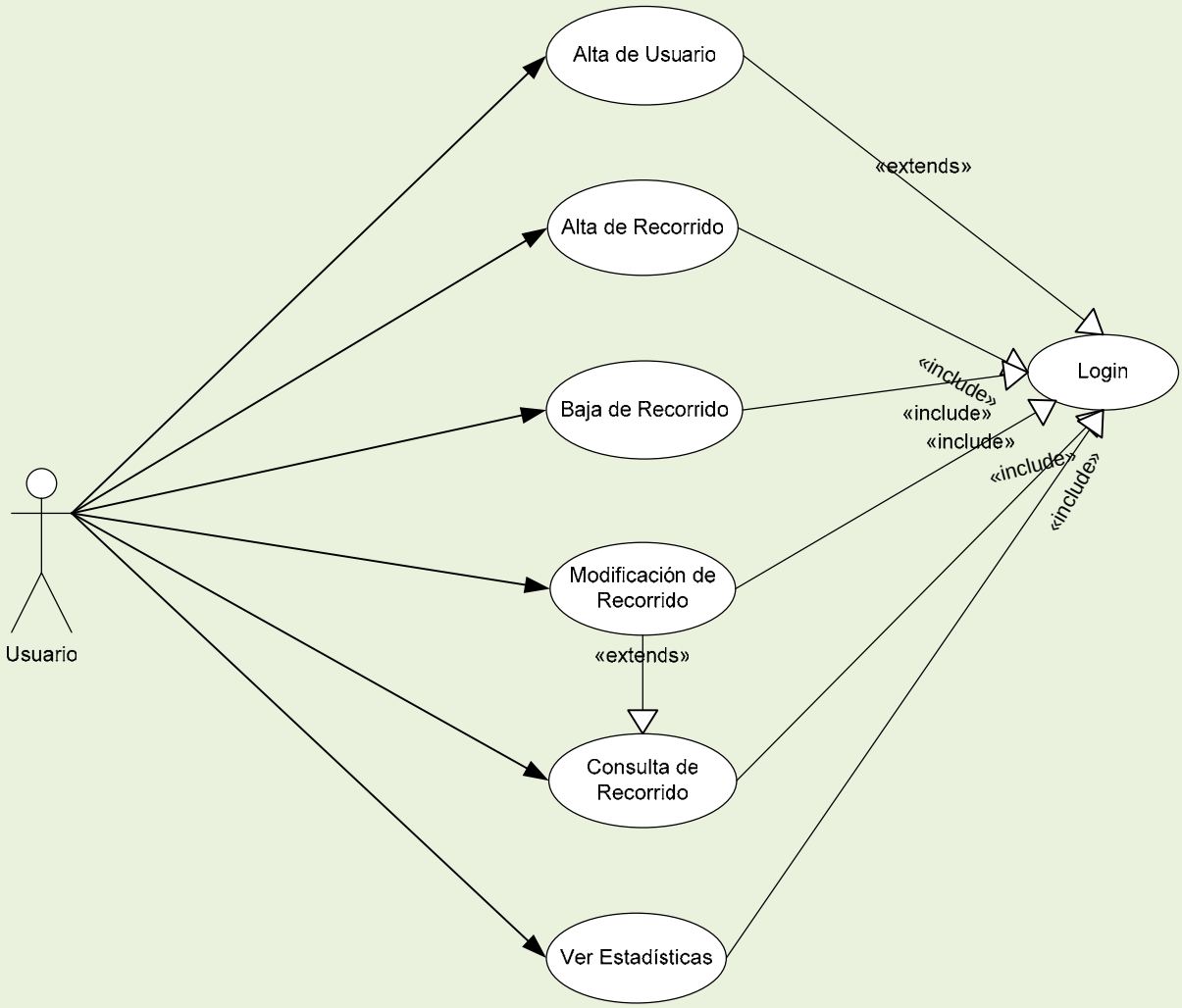


Figura 12:Casos de uso de la aplicación de ejemplo.

Cinco de los casos de uso del Usuario incluyen el caso de uso “Login” porque en algún punto de su secuencia de acontecimientos tendrán que realizar un proceso de validación de usuario. Con el objetivo de no tener que documentar cada una de las veces este proceso, se ha creado el caso

de uso que lo describe y se ha incluido desde los cinco lugares en los que aparece el proceso.

Diseño de la aplicación ejemplo: GolfStat

Introducción

El estándar JEE permite construir aplicaciones multicapa basadas en componentes utilizando el lenguaje de programación Java. Está formada por un conjunto de componentes modulares y estandarizados que ofrecen servicios automatizados para la construcción de aplicaciones distribuidas, con una estructura dividida en capas.

En nuestro caso utilizamos el framework de presentación JavaServer que utiliza una arquitectura MVC (Model-View-Controller). En este tipo de arquitecturas, el controlador se encarga de administrar el sistema de navegación, el modelo se encarga de gestionar los datos y una o varias vistas presentan los datos que almacena el modelo.

Diagrama de clases

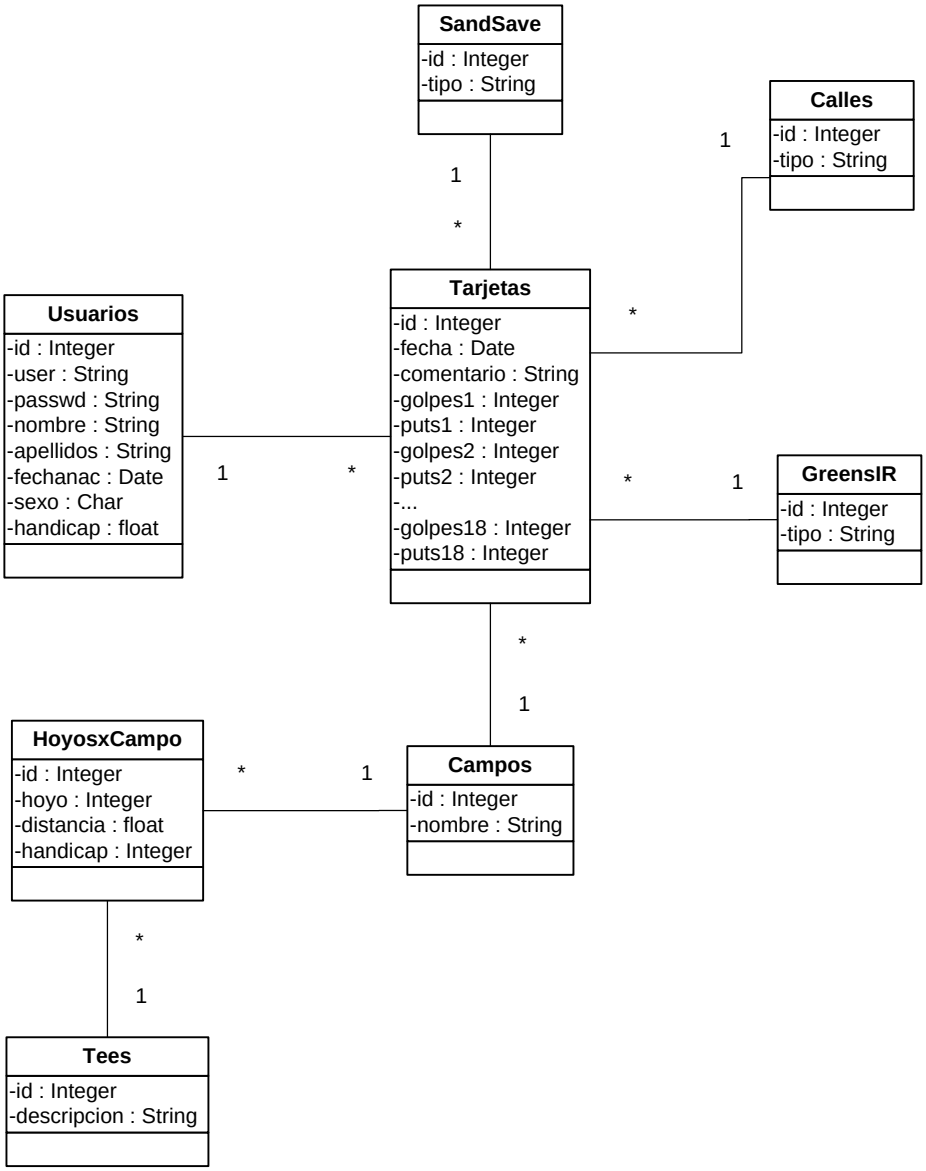


Figura 13:Diagrama de Clases de la aplicación ejemplo.

Diagramas de colaboración

Para completar el análisis pasamos a documentar el comportamiento del sistema para cada uno de los casos de uso definidos. Trataremos de analizar los mensajes que actores y sistemas intercambian.

Alta de Usuario

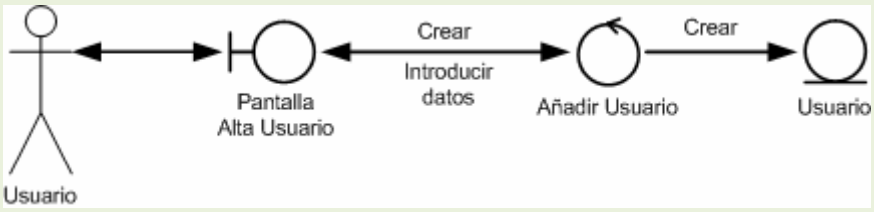


Figura 14:Diagrama de colaboración Alta de Usuario.

Alta de Recorrido

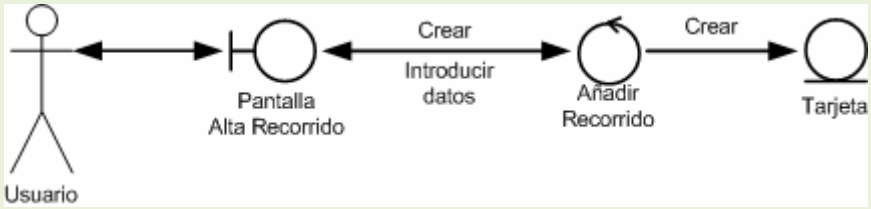


Figura 15:Diagrama de colaboración Alta de Recorrido.

Baja de Recorrido

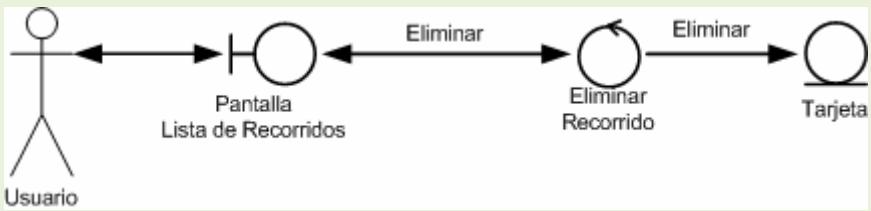


Figura 16:Diagrama de colaboración Baja de Recorrido.

Modificación de Recorrido

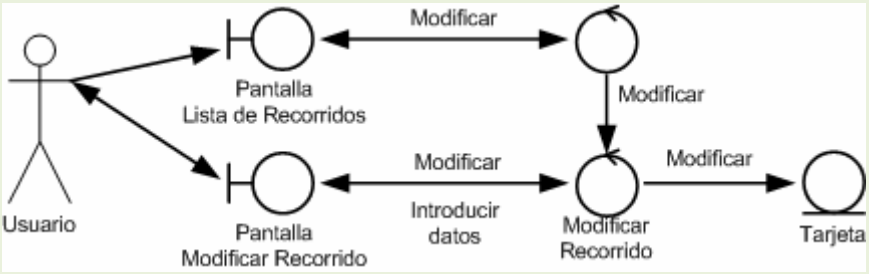


Figura 17:Diagrama de colaboración Modificación de Recorrido.

Consulta de Recorrido

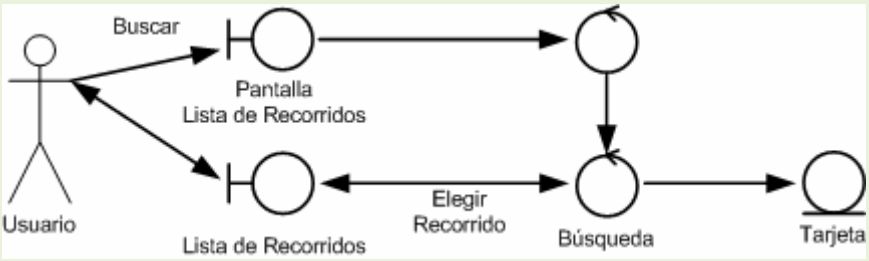


Figura 18:Diagrama de colaboración Consulta de Recorrido.

Ver Estadísticas

Figura 19:Diagrama de colaboración Ver Estadísticas.

Login

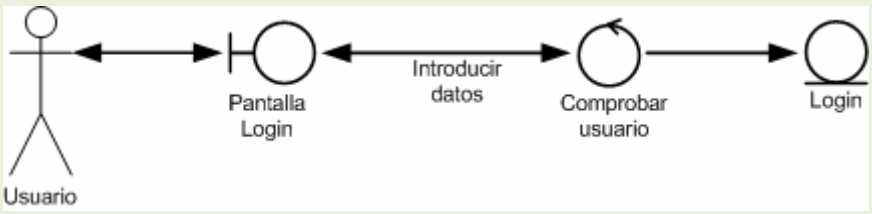


Figura 20:Diagrama de colaboración Login.

Diseño de la persistencia

Para realizar el diseño de la persistencia de la aplicación vamos primero a definir el Modelo Lógico de datos para posteriormente diseñar el modelo físico en función del Sistema de Gestión de Base de Datos utilizado en la implementación del sistema.

Nuestro modelo lógico de datos es el siguiente:

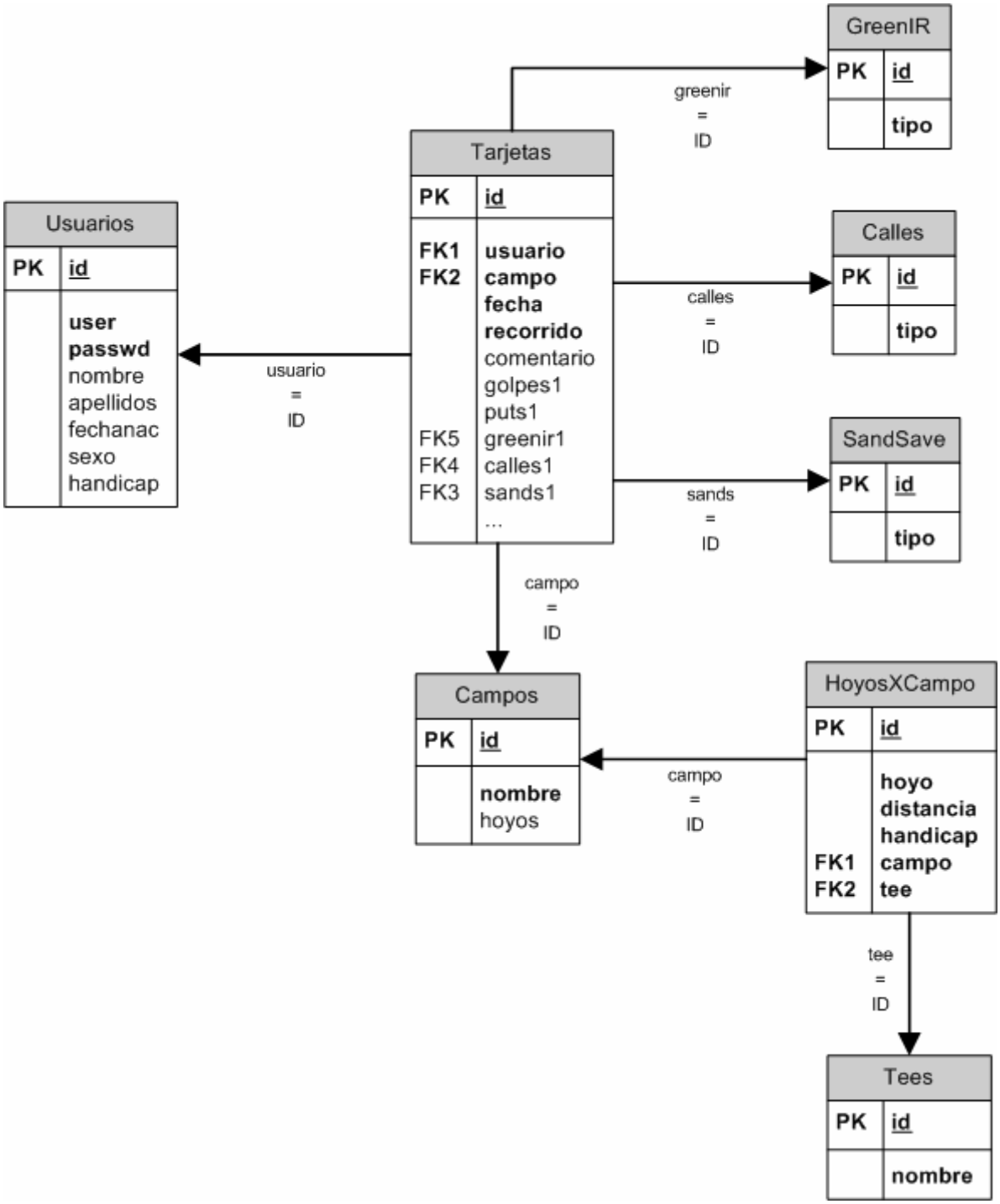


Figura 21:Modelo lógico de datos.

Hemos decidido utilizar MySQL como sistema gestor de base de datos, debido a que es uno de los más empleados en el desarrollo de aplicaciones web, además de ser una aplicación de software libre (Open Source).

Cabe destacar que en nuestro caso se producirá una baja concurrencia en la actualización de datos y será el sistema de lectura el que tendrá un uso intensivo, por lo que hace a MySQL la elección perfecta.

En el anexo A pueden consultarse los scripts de creación de las tablas del sistema.

Implementación

Desde el punto de vista de la implementación, debido a que la aplicación se diseña única y exclusivamente como ejemplo de demostración del uso del componente diseñado, se ha realizado una implementación parcial de la misma.

Dejándose fuera de nuestro estudio todos los aspectos de la aplicación que no tienen relación directa con nuestro componente, como pueden ser: la seguridad de usuario, CRUD de la mayoría de las tablas (sólo se implementan los imprescindibles para la introducción de datos), y gran parte de la operatividad de la misma.

En líneas generales la aplicación ha sido desarrollada utilizando los siguientes frameworks:

- JavaServer Faces: Capa de presentación
- Spring: Capa de negocio
- Hibernate: Capa de persistencia

Merece especial atención la implementación del uso del componente, utilizando la librería **UOComponent.jar**, creada en este proyecto.

Con el fin de mostrar las diferentes formas de uso del componente, se ha implementado el proceso de toma de datos de dos maneras distintas, una directamente desde la propia página JSP y otra a través de Beans de la capa de negocio.

En el primer caso, puede observarse cómo podemos realizar todo el proceso desde la propia página JSP:

El primer paso que debemos realizar es el de crear la estructura de datos y definir la serie a representar, para enlazarla con un Binding que la etiqueta pueda recoger y representar.

```
<%
    FacesContext context = FacesContext.getCurrentInstance();
    ValueBinding binding =
           (ValueBinding)context.getApplication().createValueBinding(
           "#{sessionScope.items}");
    FlashChartDataModel m = new FlashChartDataModel();

    m.setId(1);
    m.setDatos(Arrays.asList(new Double[]{new Double(25.0), new
           Double(50.0), new Double(20.0), new Double(5.0)}));
    m.setSeries(new FlashChartSeries(1, "pie", "50", "#FFCC33",
           "{font-size: 12px; color: #404040;}"));

    FlashChartDatos items = new FlashChartDatos();
    items.add(m);
    binding.setValue(context, items);
%>
```

Y para utilizar nuestro componente, basta con usar la etiqueta definida para tal efecto. Dentro de la etiqueta especificaremos los atributos que creamos convenientes, teniendo en cuenta que los datos a representar serán recogidos por nuestro componente desde el binding creado anteriormente.

```
<jgarciadez:fchart datos="#{items}" titulo="Título de ejemplo,
{font-size: 20px; color: #736AFF}" ancho="500" alto="500"
xleyenda="Rótulo eje X, 32,  #736AFF" />
```

En el segundo caso, podemos crear un Bean en la capa de negocio que realice las operaciones necesarias para guardar la estructura de datos y las series a representar.

En este caso los datos se recogen de la capa DAO para pasar a representarlos mediante nuestro componente

```
public class graficaBean{
    private Integer campoId;
    private Integer tarjetaId;
    private Integer usuarioId;
    private FlashChartDatos items = new FlashChartDatos();
    private FlashChartDataModel model = new FlashChartDataModel();

public String addDatos(){
    FacesContext context = FacesContext.getCurrentInstance();
    ValueBinding binding =
           (ValueBinding)context.getApplication().createValueBinding(
           "#{sessionScope.items}");
    UsuariosDAO usuariosDAO = (UsuariosDAO)ServiceFinder.findBean(
           "usuariosDao");
    TarjetasDAO dao =(TarjetasDAO)ServiceFinder.findBean(
           "tarjetasDao");
```

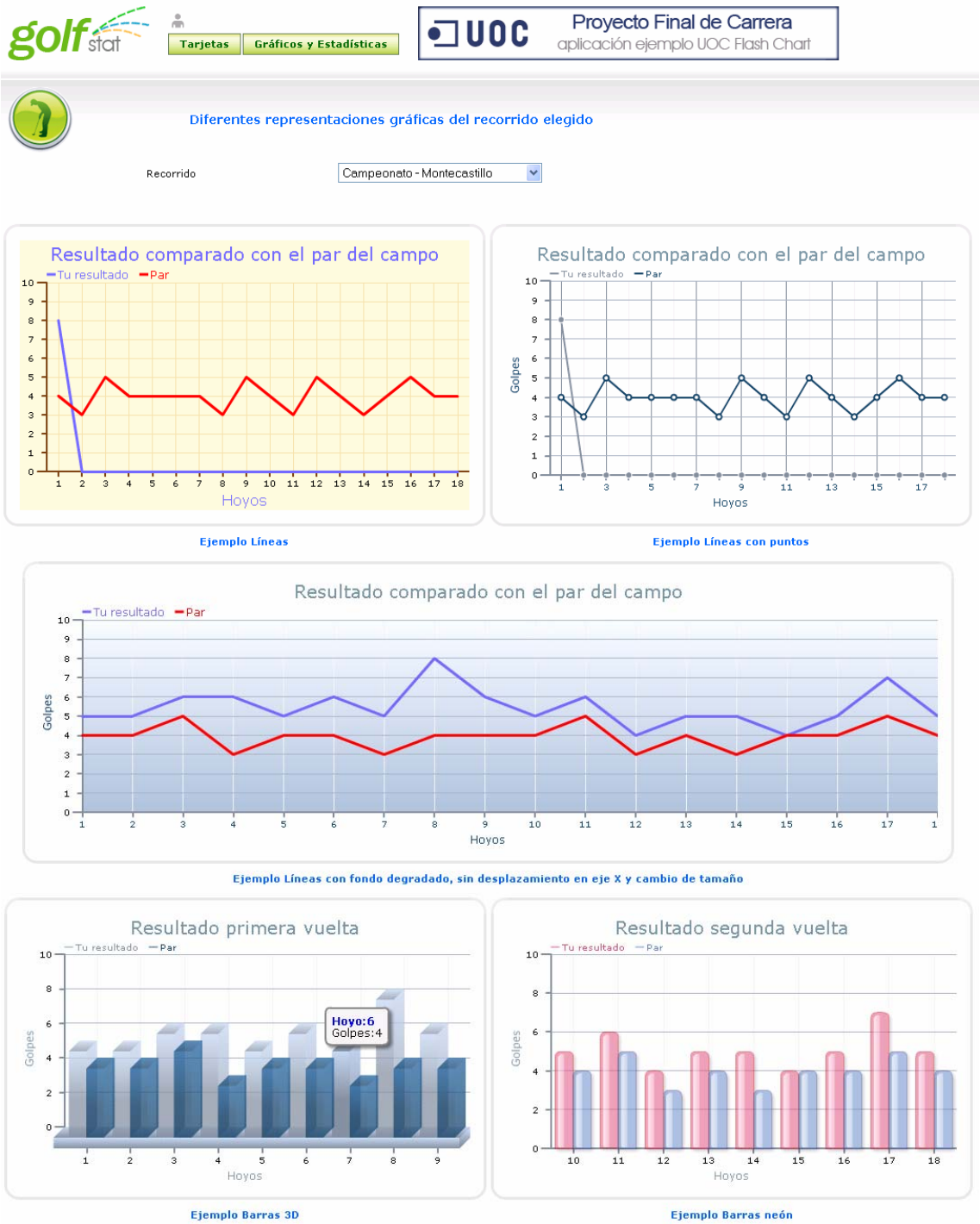
```
        usuarioId = (Integer)FacesUtil.getSessionMapValue("
                CheckValidUser.userId");
        tarjetaId = (Integer)FacesUtil.getSessionMapValue("
                graficaBean.tarjetaId");
        Tarjetas tarjeta = dao.findById(tarjetaId);

        model.setId(1);
        model.setDatos(Arrays.asList(new Integer[]{
                        tarjetaGolpes1,
                        tarjetaGolpes2,
                        tarjetaGolpes3,
                        tarjetaGolpes4,
                        tarjetaGolpes5,
                        tarjetaGolpes6,
                        tarjetaGolpes7,
                        tarjetaGolpes8,
                        tarjetaGolpes9,
                        tarjetaGolpes10,
                        tarjetaGolpes11,
                        tarjetaGolpes12,
                        tarjetaGolpes13,
                        tarjetaGolpes14,
                        tarjetaGolpes15,
                        tarjetaGolpes16,
                        tarjetaGolpes17,
                        tarjetaGolpes18}));

        model.setSeries(new FlashChartSeries(1, "line", "3",
                "#736AFF","Tu resultado", "12"));

        items.add(model);
        binding.setValue(context, items);
```

La ejecución de ambos métodos de representación, nos permitirá obtener gráficas similares a la de los siguientes ejemplos:


golf stat
Tarjetas
Gráficos y Estadísticas
UOC
Proyecto Final de Carrera
aplicación ejemplo UOC Flash Chart
Diferentes representaciones gráficas del recorrido elegido
Recorrido
Campeonato - Montecastillo
Resultado comparado con el par del campo
Tu resultado
Par
Hoyos
Ejemplo Líneas
Resultado comparado con el par del campo
Tu resultado
Par
Golpes
Hoyos
Ejemplo Líneas con puntos
Resultado comparado con el par del campo
Tu resultado
Par
Golpes
Hoyos
Ejemplo Líneas con fondo degradado, sin desplazamiento en eje X y cambio de tamaño
Resultado primera vuelta
Tu resultado
Par
Golpes
Hoyos
Hoyo:6
Golpes:4
Ejemplo Barras 3D
Resultado segunda vuelta
Tu resultado
Par
Golpes
Hoyos
Ejemplo Barras neón

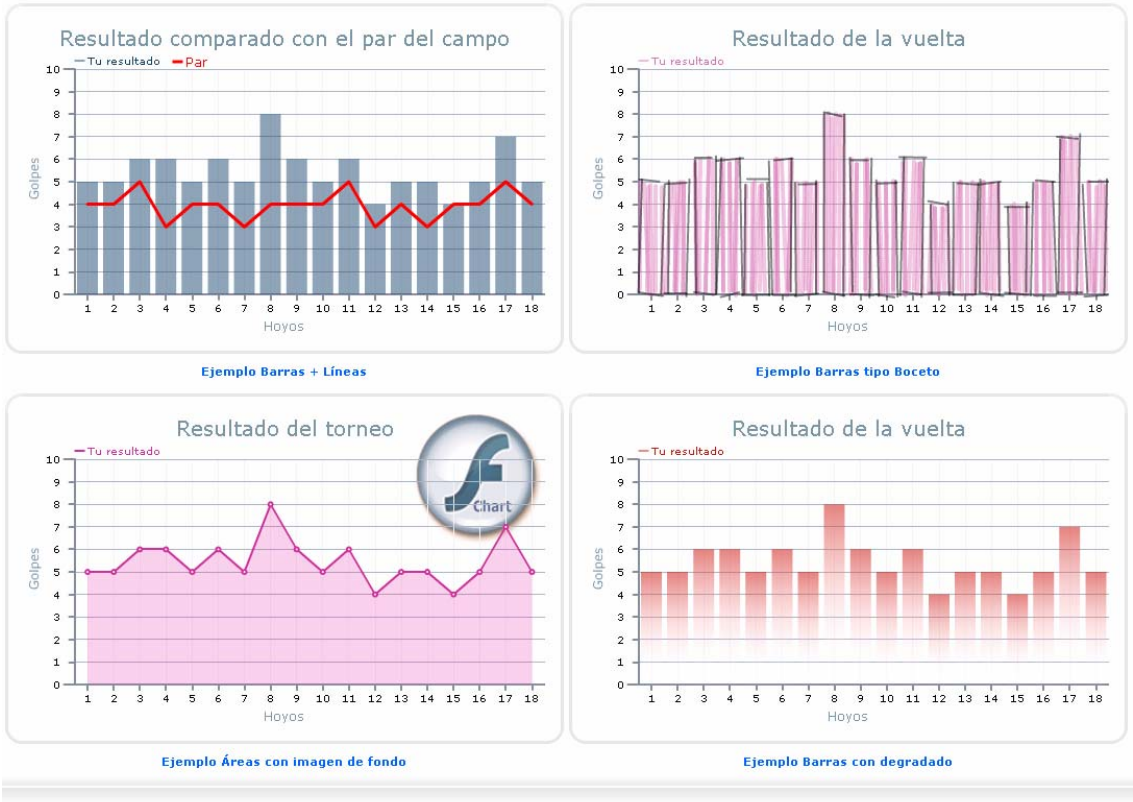


Figura 22:Pantallas de la aplicación ejemplo. GolfStat

Capítulo 6: Conclusiones

Conclusiones

Extraigo las siguientes conclusiones, una vez terminado el Proyecto, teniendo en cuenta de dónde parten mis conocimientos sobre las tecnologías utilizadas.

Por lo que creo conveniente comentar que respecto a JEE, el único contacto que había tenido con esta tecnología había sido a través de las prácticas realizadas en las asignaturas de Ingeniería del Software de Componentes y Sistemas Distribuidos (ISCSD) y Arquitectura de Sistemas Distribuidos (ASD). En las que no utilicé ningún Framework para la realización de las mismas.

La primera parte del proyecto, la fase de estudio y comparación de los Frameworks, me ha permitido asimilar los conceptos necesarios para poder afrontar la fase de diseño del componente con garantías de éxito.

El segundo bloque, el análisis, diseño e implementación del componente para el Framework JavaServer Faces, es la más importante del proyecto y la que más tiempo ha requerido para su realización. Me ha permitido aprender cuál el modelo de componentes de JSF, destacando las etapas de Encoding and Decoding Data.

Para concluir, me gustaría destacar la cantidad de tecnologías, muchas de ellas desconocidas por mi hasta la fecha, que he tenido que utilizar para la realización del proyecto: JEE, JSF, Spring, Hibernate, MySQL, Eclipse, Ant, JSP,...

Ha sido un trabajo complejo que ha llegado incluso, en algunos momentos, a desbordar mi capacidad de asimilación de conceptos, pero muy reconfortante finalmente, al comprobar que he sido capaz de afrontarlo.

Mejoras

En cuanto a las posibles mejoras que se podrían proponer, cabría destacar las siguientes:

- Desarrollar el componente para otros Frameworks.
- Ampliar los formatos de representación del componente.
- Mejorar la integración con el IDE Eclipse.

Glosario

ActionScript

Lenguaje de programación orientado a objetos, utilizado en aplicaciones web animadas realizadas en el entorno Adobe Flash.

Actor

Agente externo al sistema que interacciona en los diferentes casos de uso de la aplicación.

Adobe Flash

Es una aplicación en forma de estudio de animación que trabaja sobre "fotogramas" destinado a la producción y entrega de contenido interactivo, independiente de la plataforma de ejecución.

AJAX

Acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas.

Ant

Herramienta usada en programación para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción.

API

Acrónimo de Application Programming Interface (Interfaz de Programación de Aplicaciones) es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

ASF

Apache Software Foundation, es una organización no lucrativa, creada para dar soporte a los proyectos de software bajo la denominación Apache.

AVM

Acrónimo de ActionScript Virtual Machine (Máquina Virtual de ActionScript)

Caso de Uso

Entidad que describe una secuencia de eventos protagonizados por un actor del sistema. Esto provocará una serie de acciones y por tanto un resultado observable.

DAO

Data Access Object (Objeto de Acceso a Datos) es un componente software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos.

EJB

Acrónimo de Enterprise JavaBeans, son una de las API que forman parte del estándar de construcción de aplicaciones empresariales JEE.

Flash

Véase “Adobe Flash”

Framework

Marco de trabajo. Es un conjunto de clases o componentes utilizadas conjuntamente para dar solución a un determinado problema software.

GIF

Acrónimo de Graphics Interchange Format es un formato gráfico utilizado ampliamente en Internet, tanto para imágenes como para animaciones.

GoF

Acrónimo de Gang of Four (Banda de los Cuatro), formada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, quienes en 1995 publicaron el libro “Design Patterns”, que se convirtió en la guía de referencia del diseño de patrones.

GUI

Graphical User Interface, la interfaz gráfica de usuario es el artefacto tecnológico de un sistema interactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático.

HTML

Siglas de HyperText Markup Language (Lenguaje de Marcas de Hipertexto), es el lenguaje de marcado predominante para la construcción de páginas web.

IDE

Un entorno de desarrollo integrado, en inglés Integrated Development Environment, es un programa compuesto por un conjunto de herramientas para un programador.

JavaServer Faces

Es un framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE

JPEG

Acrónimo de Joint Photographic Experts Group, es un algoritmo diseñado para comprimir imágenes con 24 bits de profundidad o en escala de grises. JPEG es también el formato de archivo que utiliza este algoritmo para almacenar las imágenes comprimidas.

JSF

Véase JavaServer Faces.

JSP

JavaServer Pages es una tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo.

Model-View-Controller

Es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

MVC

Acrónimo de Model-View-Controller.

Open source

Código abierto, en inglés open source, es el término con el que se conoce al software distribuido y desarrollado libremente.

Patrón

El concepto general en el que se basan los patrones es el de ofrecer una solución experta a un problema común.

PNG

Acrónimo de Portable Network Graphics, es un formato gráfico basado en un algoritmo de compresión sin pérdida para bitmaps no sujeto a patentes.

RAD

Rapid application development, es un proceso de desarrollo de software, que comprende el desarrollo interactivo, la construcción de prototipos y el uso de herramientas CASE (Computer Aided Software Engineering).

Spring

Es un framework de código abierto de desarrollo de aplicaciones para la plataforma Java.

Struts

Struts es un marco de trabajo de soporte para el desarrollo de aplicaciones Web bajo el patrón MVC bajo la plataforma JEE.

TLD

Acrónimo de Tag Library Descriptor.

Tomcat

Funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation.

UI

Acrónimo de User Interface (Interfaz de Usuario).

URL

Siglas de Uniform Resource Locator (localizador uniforme de recurso). Es una secuencia de caracteres, de acuerdo a un formato estándar, que se usa para nombrar recursos, como documentos e imágenes en Internet, para su localización.

XML

Extensible Markup Language (Lenguaje de marcas extensible). Es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium.

Bibliografía

Libros

Spring in Action, Second Edition - Craig Walls with Ryan Breidenbach, August, 2007. ISBN: 1-933988-13-4

Java Persistence with Hibernate - Christian Bauer and Gavin King, November, 2006. ISBN: 1-932394-88-5

JavaServer Faces in Action - Kito D. Mann, 2004. ISBN: 1932394125

Struts in Action – Ted Husted, 2003. ISBN: 1930110502

La Biblia de Java 2 - Schildt, Herbert. ISBN: 978-84-415-1865-0

Tutoriales

http://www.exadel.com/web/portal/products/Tutorials:

- JSF: Getting Started Guide for Creating a JSF Application.
- JSF: JSF KickStart- A Simple JavaServer Faces Application.
- JSF: How To Write Your Own JSF Components.
- JSF: Java Server Faces HTML Tags.
- JSF: A JSF-Based GuessNumber Application.

Internet

Adictos al Trabajo

http://www.adictosaltrabajo.com

Api4 Java

http://www.api4java.net

Desarrollo Web

http://www.desarrolloweb.com

Eclipse

http://www.eclipse.com

Java Hispano

http://www.javahispano.org

JBoss

http://www.jboss.org

JGuru

http://www.jguru.com

JSF Central

http://www.jsfcentral.com

JSF Community

https://javaserverfaces.dev.java.net

JSF Resources- Oracle

http://www.oracle.com/technology/tech/java/jsf.html

MySQL

http://www.mysql.com

Open Flash Chart

http://teethgrinder.co.uk/open-flash-chart/

Programación en Castellano

http://www.programacion.com

Struts Jakarta

http://struts.apache.org

SUN

http://java.sun.com

The Apache Jakarta Projetc

http://jakarta.apache.org

Tomcat

http://jakarta.apache.org

Anexos

Anexo A

Scripts de creación de las tablas de la aplicación de ejemplo: GolfStat

Creación de la Base de Datos

```
CREATE DATABASE `golfstat`;
```

Calles

```
DROP TABLE IF EXISTS `golfstat`.`calles`;
CREATE TABLE  `golfstat`.`calles` (
  `id` int(10) unsigned NOT NULL auto_increment COMMENT 'Clave principal',
  `tipo` varchar(45) NOT NULL COMMENT 'Tipo de salida en calle',
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Tabla de tipos de salida en
calle';
```

Campos

```
DROP TABLE IF EXISTS `golfstat`.`campos`;
CREATE TABLE  `golfstat`.`campos` (
  `id` int(10) unsigned NOT NULL auto_increment COMMENT 'Clave principal',
  `nombre` varchar(255) NOT NULL COMMENT 'Nombre del campo de golf',
  `descripcion` varchar(255) default NULL COMMENT 'Descripcion del campo',
  `hoyos` int(10) unsigned NOT NULL COMMENT 'Numero de hoyos que tiene el
campo',
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Tabla de campos de golf
disponibles en la aplicación';
```

Greenir

```
DROP TABLE IF EXISTS `golfstat`.`greenir`;
CREATE TABLE  `golfstat`.`greenir` (
  `id` int(10) unsigned NOT NULL auto_increment COMMENT 'Clave principal',
  `tipo` varchar(45) NOT NULL COMMENT 'Tipo de green en regulacion',
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Tabla de tipos de posicion
de Green in Regulation';
```

Hoyosxcampo

```
DROP TABLE IF EXISTS `golfstat`.`hoyosxcampo`;
CREATE TABLE  `golfstat`.`hoyosxcampo` (
  `id` int(10) unsigned NOT NULL auto_increment COMMENT 'Clave principal',
  `camposid` int(10) unsigned NOT NULL COMMENT 'Codigo del campo',
  `teesid` int(10) unsigned NOT NULL COMMENT 'Codigo del tipo de tee de
salida',
  `hoyo` int(10) unsigned NOT NULL COMMENT 'Numero de hoyo',
  `distancia` int(10) unsigned NOT NULL COMMENT 'Distancia del hoyo en
metros',
  `handicap` int(10) unsigned NOT NULL COMMENT 'Handicap del hoyo',
  `par` int(10) unsigned NOT NULL,
```

```
  PRIMARY KEY  (`id`),
  KEY `FK_hoyosxcampo_1` (`camposid`),
  CONSTRAINT  `FK_hoyosxcampo_1`  FOREIGN  KEY  (`camposid`)  REFERENCES
`campos` (`id`),
  CONSTRAINT `FK_hoyosxcampo_2` FOREIGN KEY (`camposid`) REFERENCES `tees`
(`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Tabla de características de
los hoyos de cada campo';
```

Sandsave

```
DROP TABLE IF EXISTS `golfstat`.`sandsave`;
CREATE TABLE  `golfstat`.`sandsave` (
  `id` int(10) unsigned NOT NULL auto_increment COMMENT 'Clave principal',
  `tipo` varchar(45) NOT NULL COMMENT 'Tipo de salida de bunker',
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Tabla de tipos de salida de
bunkers';
```

Tarjetas

```
DROP TABLE IF EXISTS `golfstat`.`tarjetas`;
CREATE TABLE  `golfstat`.`tarjetas` (
  `id` int(10) unsigned NOT NULL auto_increment COMMENT 'Clave principal',
  `usuarioid` int(10) unsigned NOT NULL,
  `camposid` int(10) unsigned NOT NULL COMMENT 'Codigo del campo',
  `teesid` int(10) unsigned NOT NULL COMMENT 'Codigo del tipo de tee  de
salida',
  `fecha` datetime NOT NULL COMMENT 'Fecha del recorrido',
  `recorrido` varchar(100) NOT NULL COMMENT 'Nombre del recorrido',
  `comentario` varchar(255) default NULL COMMENT 'Cometarios anotados por
el usuario',
  `golpes1` int(10) unsigned default NULL COMMENT 'Golpes Hoyo 1',
  `puts1` int(10) unsigned default NULL COMMENT 'Puts Hoyo 1',
  `callesid1` int(10) unsigned default NULL COMMENT 'Fairway Hit Hoyo 1',
  `girid1` int(10) unsigned default NULL COMMENT 'Green In Regulation Hoyo
1',
  `ssid1` int(10) unsigned default NULL COMMENT 'Sand Save Hoyo 1',
  `golpes2` int(10) unsigned default NULL,
  `puts2` int(10) unsigned default NULL,
  `callesid2` int(10) unsigned default NULL,
  `girid2` int(10) unsigned default NULL,
  `ssid2` int(10) unsigned default NULL,
  `golpes3` int(10) unsigned default NULL,
  `puts3` int(10) unsigned default NULL,
  `callesid3` int(10) unsigned default NULL,
  `girid3` int(10) unsigned default NULL,
  `ssid3` int(10) unsigned default NULL,
  `golpes4` int(10) unsigned default NULL,
  `puts4` int(10) unsigned default NULL,
  `callesid4` int(10) unsigned default NULL,
  `girid4` int(10) unsigned default NULL,
  `ssid4` int(10) unsigned default NULL,
  `golpes5` int(10) unsigned default NULL,
  `puts5` int(10) unsigned default NULL,
  `callesid5` int(10) unsigned default NULL,
  `girid5` int(10) unsigned default NULL,
  `ssid5` int(10) unsigned default NULL,
  `golpes6` int(10) unsigned default NULL,
  `puts6` int(10) unsigned default NULL,
  `callesid6` int(10) unsigned default NULL,
  `girid6` int(10) unsigned default NULL,
  `ssid6` int(10) unsigned default NULL,
  `golpes7` int(10) unsigned default NULL,
```

```
`puts7` int(10) unsigned default NULL,
`callesid7` int(10) unsigned default NULL,
`girid7` int(10) unsigned default NULL,
`ssid7` int(10) unsigned default NULL,
`golpes8` int(10) unsigned default NULL,
`puts8` int(10) unsigned default NULL,
`callesid8` int(10) unsigned default NULL,
`girid8` int(10) unsigned default NULL,
`ssid8` int(10) unsigned default NULL,
`golpes9` int(10) unsigned default NULL,
`puts9` int(10) unsigned default NULL,
`callesid9` int(10) unsigned default NULL,
`girid9` int(10) unsigned default NULL,
`ssid9` int(10) unsigned default NULL,
`golpes10` int(10) unsigned default NULL,
`puts10` int(10) unsigned default NULL,
`callesid10` int(10) unsigned default NULL,
`girid10` int(10) unsigned default NULL,
`ssid10` int(10) unsigned default NULL,
`golpes11` int(10) unsigned default NULL,
`puts11` int(10) unsigned default NULL,
`callesid11` int(10) unsigned default NULL,
`girid11` int(10) unsigned default NULL,
`ssid11` int(10) unsigned default NULL,
`golpes12` int(10) unsigned default NULL,
`puts12` int(10) unsigned default NULL,
`callesid12` int(10) unsigned default NULL,
`girid12` int(10) unsigned default NULL,
`ssid12` int(10) unsigned default NULL,
`golpes13` int(10) unsigned default NULL,
`puts13` int(10) unsigned default NULL,
`callesid13` int(10) unsigned default NULL,
`girid13` int(10) unsigned default NULL,
`ssid13` int(10) unsigned default NULL,
`golpes14` int(10) unsigned default NULL,
`puts14` int(10) unsigned default NULL,
`callesid14` int(10) unsigned default NULL,
`girid14` int(10) unsigned default NULL,
`ssid14` int(10) unsigned default NULL,
`golpes15` int(10) unsigned default NULL,
`puts15` int(10) unsigned default NULL,
`callesid15` int(10) unsigned default NULL,
`girid15` int(10) unsigned default NULL,
`ssid15` int(10) unsigned default NULL,
`golpes16` int(10) unsigned default NULL,
`puts16` int(10) unsigned default NULL,
`callesid16` int(10) unsigned default NULL,
`girid16` int(10) unsigned default NULL,
`ssid16` int(10) unsigned default NULL,
`golpes17` int(10) unsigned default NULL,
`puts17` int(10) unsigned default NULL,
`callesid17` int(10) unsigned default NULL,
`girid17` int(10) unsigned default NULL,
`ssid17` int(10) unsigned default NULL,
`golpes18` int(10) unsigned default NULL,
`puts18` int(10) unsigned default NULL,
`callesid18` int(10) unsigned default NULL,
`girid18` int(10) unsigned default NULL,
`ssid18` int(10) unsigned default NULL,
PRIMARY KEY  (`id`),
KEY `FK_tarjetas_1` (`camposid`),
KEY `FK_tarjetas_3` (`teesid`),
KEY `FK_tarjetas_4` (`callesid1`),
KEY `FK_tarjetas_5` (`girid1`),
KEY `FK_tarjetas_6` (`ssid1`),
KEY `FK_tarjetas_2` (`usuarioid`),
KEY `FK_tarjetas_7` (`callesid2`),
KEY `FK_tarjetas_8` (`girid2`),
```

```
  KEY `FK_tarjetas_9` (`ssid2`),
  KEY `FK_tarjetas_10` (`callesid3`),
  KEY `FK_tarjetas_11` (`girid3`),
  KEY `FK_tarjetas_12` (`ssid3`),
  KEY `FK_tarjetas_13` (`callesid4`),
  KEY `FK_tarjetas_14` (`girid4`),
  KEY `FK_tarjetas_15` (`ssid4`),
  KEY `FK_tarjetas_16` (`callesid5`),
  KEY `FK_tarjetas_17` (`girid5`),
  KEY `FK_tarjetas_18` (`ssid5`),
  KEY `FK_tarjetas_19` (`callesid6`),
  KEY `FK_tarjetas_20` (`girid6`),
  KEY `FK_tarjetas_21` (`ssid6`),
  KEY `FK_tarjetas_22` (`callesid7`),
  KEY `FK_tarjetas_23` (`girid7`),
  KEY `FK_tarjetas_24` (`ssid7`),
  KEY `FK_tarjetas_25` (`callesid8`),
  KEY `FK_tarjetas_26` (`girid8`),
  KEY `FK_tarjetas_27` (`ssid8`),
  KEY `FK_tarjetas_28` (`callesid9`),
  KEY `FK_tarjetas_29` (`girid9`),
  KEY `FK_tarjetas_30` (`ssid9`),
  KEY `FK_tarjetas_31` (`callesid10`),
  KEY `FK_tarjetas_32` (`girid10`),
  KEY `FK_tarjetas_33` (`ssid10`),
  KEY `FK_tarjetas_34` (`callesid11`),
  KEY `FK_tarjetas_35` (`girid11`),
  KEY `FK_tarjetas_36` (`ssid11`),
  KEY `FK_tarjetas_37` (`callesid12`),
  KEY `FK_tarjetas_38` (`girid12`),
  KEY `FK_tarjetas_39` (`ssid12`),
  KEY `FK_tarjetas_40` (`callesid13`),
  KEY `FK_tarjetas_41` (`girid13`),
  KEY `FK_tarjetas_42` (`ssid13`),
  KEY `FK_tarjetas_43` (`callesid14`),
  KEY `FK_tarjetas_44` (`girid14`),
  KEY `FK_tarjetas_45` (`ssid14`),
  KEY `FK_tarjetas_46` (`callesid15`),
  KEY `FK_tarjetas_47` (`girid15`),
  KEY `FK_tarjetas_48` (`ssid15`),
  KEY `FK_tarjetas_49` (`callesid16`),
  KEY `FK_tarjetas_50` (`girid16`),
  KEY `FK_tarjetas_51` (`ssid16`),
  KEY `FK_tarjetas_52` (`callesid17`),
  KEY `FK_tarjetas_53` (`girid17`),
  KEY `FK_tarjetas_54` (`ssid17`),
  KEY `FK_tarjetas_55` (`callesid18`),
  KEY `FK_tarjetas_56` (`girid18`),
  KEY `FK_tarjetas_57` (`ssid18`),
  CONSTRAINT `FK_tarjetas_1` FOREIGN KEY (`camposid`) REFERENCES `campos`
(`id`),
  CONSTRAINT `FK_tarjetas_10` FOREIGN KEY (`callesid3`) REFERENCES `calles`
(`id`),
  CONSTRAINT `FK_tarjetas_11` FOREIGN KEY (`girid3`) REFERENCES `greenir`
(`id`),
  CONSTRAINT `FK_tarjetas_12` FOREIGN KEY (`ssid3`) REFERENCES `sandsave`
(`id`),
  CONSTRAINT `FK_tarjetas_13` FOREIGN KEY (`callesid4`) REFERENCES `calles`
(`id`),
  CONSTRAINT `FK_tarjetas_14` FOREIGN KEY (`girid4`) REFERENCES `greenir`
(`id`),
  CONSTRAINT `FK_tarjetas_15` FOREIGN KEY (`ssid4`) REFERENCES `sandsave`
(`id`),
  CONSTRAINT `FK_tarjetas_16` FOREIGN KEY (`callesid5`) REFERENCES `calles`
(`id`),
  CONSTRAINT `FK_tarjetas_17` FOREIGN KEY (`girid5`) REFERENCES `greenir`
(`id`),
```

```
  CONSTRAINT `FK_tarjetas_18` FOREIGN KEY (`ssid5`) REFERENCES `sandsave`
(`id`),
  CONSTRAINT `FK_tarjetas_19` FOREIGN KEY (`callesid6`) REFERENCES `calles`
(`id`),
  CONSTRAINT `FK_tarjetas_2` FOREIGN KEY (`usuarioid`) REFERENCES
`usuarios` (`id`),
  CONSTRAINT `FK_tarjetas_20` FOREIGN KEY (`girid6`) REFERENCES `greenir`
(`id`),
  CONSTRAINT `FK_tarjetas_21` FOREIGN KEY (`ssid6`) REFERENCES `sandsave`
(`id`),
  CONSTRAINT `FK_tarjetas_22` FOREIGN KEY (`callesid7`) REFERENCES `calles`
(`id`),
  CONSTRAINT `FK_tarjetas_23` FOREIGN KEY (`girid7`) REFERENCES `greenir`
(`id`),
  CONSTRAINT `FK_tarjetas_24` FOREIGN KEY (`ssid7`) REFERENCES `sandsave`
(`id`),
  CONSTRAINT `FK_tarjetas_25` FOREIGN KEY (`callesid8`) REFERENCES `calles`
(`id`),
  CONSTRAINT `FK_tarjetas_26` FOREIGN KEY (`girid8`) REFERENCES `greenir`
(`id`),
  CONSTRAINT `FK_tarjetas_27` FOREIGN KEY (`ssid8`) REFERENCES `sandsave`
(`id`),
  CONSTRAINT `FK_tarjetas_28` FOREIGN KEY (`callesid9`) REFERENCES `calles`
(`id`),
  CONSTRAINT `FK_tarjetas_29` FOREIGN KEY (`girid9`) REFERENCES `greenir`
(`id`),
  CONSTRAINT `FK_tarjetas_3` FOREIGN KEY (`teesid`) REFERENCES `tees`
(`id`),
  CONSTRAINT `FK_tarjetas_30` FOREIGN KEY (`ssid9`) REFERENCES `sandsave`
(`id`),
  CONSTRAINT `FK_tarjetas_31` FOREIGN KEY (`callesid10`) REFERENCES
`calles` (`id`),
  CONSTRAINT `FK_tarjetas_32` FOREIGN KEY (`girid10`) REFERENCES `greenir`
(`id`),
  CONSTRAINT `FK_tarjetas_33` FOREIGN KEY (`ssid10`) REFERENCES `sandsave`
(`id`),
  CONSTRAINT `FK_tarjetas_34` FOREIGN KEY (`callesid11`) REFERENCES
`calles` (`id`),
  CONSTRAINT `FK_tarjetas_35` FOREIGN KEY (`girid11`) REFERENCES `greenir`
(`id`),
  CONSTRAINT `FK_tarjetas_36` FOREIGN KEY (`ssid11`) REFERENCES `sandsave`
(`id`),
  CONSTRAINT `FK_tarjetas_37` FOREIGN KEY (`callesid12`) REFERENCES
`calles` (`id`),
  CONSTRAINT `FK_tarjetas_38` FOREIGN KEY (`girid12`) REFERENCES `greenir`
(`id`),
  CONSTRAINT `FK_tarjetas_39` FOREIGN KEY (`ssid12`) REFERENCES `sandsave`
(`id`),
  CONSTRAINT `FK_tarjetas_4` FOREIGN KEY (`callesid1`) REFERENCES `calles`
(`id`),
  CONSTRAINT `FK_tarjetas_40` FOREIGN KEY (`callesid13`) REFERENCES
`calles` (`id`),
  CONSTRAINT `FK_tarjetas_41` FOREIGN KEY (`girid13`) REFERENCES `greenir`
(`id`),
  CONSTRAINT `FK_tarjetas_42` FOREIGN KEY (`ssid13`) REFERENCES `sandsave`
(`id`),
  CONSTRAINT `FK_tarjetas_43` FOREIGN KEY (`callesid14`) REFERENCES
`calles` (`id`),
  CONSTRAINT `FK_tarjetas_44` FOREIGN KEY (`girid14`) REFERENCES `greenir`
(`id`),
  CONSTRAINT `FK_tarjetas_45` FOREIGN KEY (`ssid14`) REFERENCES `sandsave`
(`id`),
  CONSTRAINT `FK_tarjetas_46` FOREIGN KEY (`callesid15`) REFERENCES
`calles` (`id`),
  CONSTRAINT `FK_tarjetas_47` FOREIGN KEY (`girid15`) REFERENCES `greenir`
(`id`),
  CONSTRAINT `FK_tarjetas_48` FOREIGN KEY (`ssid15`) REFERENCES `sandsave`
(`id`),
```

```
  CONSTRAINT  `FK_tarjetas_49`  FOREIGN  KEY  (`callesid16`)  REFERENCES
`calles` (`id`),
  CONSTRAINT `FK_tarjetas_5` FOREIGN KEY (`girid1`) REFERENCES `greenir`
(`id`),
  CONSTRAINT `FK_tarjetas_50` FOREIGN KEY (`girid16`) REFERENCES `greenir`
(`id`),
  CONSTRAINT `FK_tarjetas_51` FOREIGN KEY (`ssid16`) REFERENCES `sandsave`
(`id`),
  CONSTRAINT  `FK_tarjetas_52`  FOREIGN  KEY  (`callesid17`)  REFERENCES
`calles` (`id`),
  CONSTRAINT `FK_tarjetas_53` FOREIGN KEY (`girid17`) REFERENCES `greenir`
(`id`),
  CONSTRAINT `FK_tarjetas_54` FOREIGN KEY (`ssid17`) REFERENCES `sandsave`
(`id`),
  CONSTRAINT  `FK_tarjetas_55`  FOREIGN  KEY  (`callesid18`)  REFERENCES
`calles` (`id`),
  CONSTRAINT `FK_tarjetas_56` FOREIGN KEY (`girid18`) REFERENCES `greenir`
(`id`),
  CONSTRAINT `FK_tarjetas_57` FOREIGN KEY (`ssid18`) REFERENCES `sandsave`
(`id`),
  CONSTRAINT `FK_tarjetas_6` FOREIGN KEY (`ssid1`) REFERENCES `sandsave`
(`id`),
  CONSTRAINT `FK_tarjetas_7` FOREIGN KEY (`callesid2`) REFERENCES `calles`
(`id`),
  CONSTRAINT `FK_tarjetas_8` FOREIGN KEY (`girid2`) REFERENCES `greenir`
(`id`),
  CONSTRAINT `FK_tarjetas_9` FOREIGN KEY (`ssid2`) REFERENCES `sandsave`
(`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Tabla de tarjetas de los
usuarios';
```

Tees

```
DROP TABLE IF EXISTS `golfstat`.`tees`;
CREATE TABLE  `golfstat`.`tees` (
  `id` int(10) unsigned NOT NULL auto_increment COMMENT 'Clave principal',
  `descripcion` varchar(45) NOT NULL COMMENT 'Nombre del tee de salida',
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Tabla de tipos de tees de
salida disponibles en los campos';
```

Usuarios

```
DROP TABLE IF EXISTS `golfstat`.`usuarios`;
CREATE TABLE  `golfstat`.`usuarios` (
  `id` int(10) unsigned NOT NULL auto_increment COMMENT 'Clave principal',
  `user` varchar(45) character set latin1 collate latin1_spanish_ci NOT
NULL COMMENT 'Nombre de usuario',
  `passwd` varchar(25) character set latin1 collate latin1_spanish_ci NOT
NULL COMMENT 'Contraseña de acceso',
  `nombre` varchar(55) character set latin1 collate latin1_spanish_ci
default NULL COMMENT 'Nombre del usuario',
  `apellidos` varchar(76) character set latin1 collate latin1_spanish_ci
default NULL COMMENT 'Apelidos del usuario',
  `fechanac` datetime default NULL COMMENT 'Fecha de nacimiento',
  `sexo` varchar(1) default NULL COMMENT 'Sexo del usuario',
  `handicap` float default NULL COMMENT 'Handicap del usuario',
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Tabla de usuarios';
```

Anexo B

Scripts para la inserción de registros de ejemplo en la base de datos.

Calles

```
INSERT INTO `calles` (`id`, `tipo`) VALUES
(1, '---'),
(2, 'Sí'),
(3, 'No'),
(4, 'Centro'),
(5, 'Izquierda'),
(6, 'Derecha'),
(7, 'Corto'),
(8, 'Rough (I)'),
(9, 'Rough (D)');
```

Campos

```
INSERT INTO `campos` (`id`, `nombre`, `descripcion`, `hoyos`) VALUES
(1, 'Sherry Golf Jerez', NULL, 18),
(2, 'Montecastillo', NULL, 18);
```

Greenir

```
INSERT INTO `greenir` (`id`, `tipo`) VALUES
(1, '---'),
(2, 'Sí'),
(3, 'No'),
(4, 'Corto'),
(5, 'Largo'),
(6, 'Izquierda'),
(7, 'Derecha');
```

Hoyosxcampo

```
INSERT INTO `hoyosxcampo` (`id`, `camposid`, `teesid`, `hoyo`, `distancia`,
`handicap`, `par`) VALUES
(1, 1, 1, 1, 363, 2, 4),
(2, 1, 1, 2, 384, 13, 4),
(3, 1, 1, 3, 477, 14, 5),
(4, 1, 1, 4, 156, 8, 3),
(5, 1, 1, 5, 498, 9, 4),
(6, 1, 1, 6, 319, 15, 4),
(7, 1, 1, 7, 150, 17, 3),
(8, 1, 1, 8, 399, 3, 4),
(9, 1, 1, 9, 390, 7, 4),
(10, 1, 1, 10, 333, 10, 4),
(11, 1, 1, 11, 497, 5, 5),
(12, 1, 1, 12, 159, 18, 3),
(13, 1, 1, 13, 424, 1, 4),
(14, 1, 1, 14, 127, 11, 3),
(15, 1, 1, 15, 382, 6, 4),
(16, 1, 1, 16, 240, 12, 4),
(17, 1, 1, 17, 449, 16, 5),
(18, 1, 1, 18, 413, 4, 4),
(19, 1, 2, 1, 269, 2, 3),
(20, 1, 2, 2, 305, 13, 4),
(21, 1, 2, 3, 384, 14, 5),
(22, 1, 2, 4, 114, 8, 3),
(23, 1, 2, 5, 459, 9, 4),
```

```
(24, 1, 2, 6, 221, 15, 4),
(25, 1, 2, 7, 111, 17, 3),
(26, 1, 2, 8, 296, 3, 4),
(27, 1, 2, 9, 318, 7, 4),
(28, 1, 2, 10, 242, 10, 4),
(29, 1, 2, 11, 425, 5, 5),
(30, 1, 2, 12, 96, 18, 3),
(31, 1, 2, 13, 328, 1, 4),
(32, 1, 2, 14, 105, 11, 3),
(33, 1, 2, 15, 296, 6, 4),
(34, 1, 2, 16, 200, 12, 4),
(35, 1, 2, 17, 375, 16, 5),
(36, 1, 2, 18, 323, 4, 4),
(37, 2, 1, 1, 293, 17, 4),
(38, 2, 1, 2, 163, 11, 3),
(39, 2, 1, 3, 494, 3, 5),
(40, 2, 1, 4, 326, 12, 4),
(41, 2, 1, 5, 353, 7, 4),
(42, 2, 1, 6, 352, 13, 4),
(43, 2, 1, 7, 370, 9, 4),
(44, 2, 1, 8, 161, 8, 3),
(45, 2, 1, 9, 462, 4, 5),
(46, 2, 1, 10, 354, 5, 4),
(47, 2, 1, 11, 196, 14, 3),
(48, 2, 1, 12, 464, 15, 5),
(49, 2, 1, 13, 372, 2, 4),
(50, 2, 1, 14, 142, 18, 3),
(51, 2, 1, 15, 402, 1, 4),
(52, 2, 1, 16, 453, 16, 5),
(53, 2, 1, 17, 313, 6, 4),
(54, 2, 1, 18, 373, 10, 4),
(55, 2, 2, 1, 254, 17, 4),
(56, 2, 2, 2, 129, 11, 3),
(57, 2, 2, 3, 436, 3, 5),
(58, 2, 2, 4, 282, 12, 4),
(59, 2, 2, 5, 318, 7, 4),
(60, 2, 2, 6, 285, 13, 4),
(61, 2, 2, 7, 289, 9, 4),
(62, 2, 2, 8, 123, 8, 3),
(63, 2, 2, 9, 416, 4, 5),
(64, 2, 2, 10, 298, 5, 4),
(65, 2, 2, 11, 166, 14, 3),
(66, 2, 2, 12, 427, 15, 5),
(67, 2, 2, 13, 340, 2, 4),
(68, 2, 2, 14, 119, 18, 3),
(69, 2, 2, 15, 352, 1, 4),
(70, 2, 2, 16, 422, 16, 5),
(71, 2, 2, 17, 255, 6, 4),
(72, 2, 2, 18, 319, 10, 4);
```

Sandsave

```
INSERT INTO `sandsave` (`id`, `tipo`) VALUES
(1, '---'),
(2, 'Sí'),
(3, 'No'),
(4, 'Fallado');
```

Tarjetas

```
INSERT INTO `tarjetas` (`id`, `usuarioid`, `camposid`, `teesid`, `fecha`,
`recorrido`, `comentario`, `golpes1`, `puts1`, `callesid1`, `girid1`,
`ssid1`, `golpes2`, `puts2`, `callesid2`, `girid2`, `ssid2`, `golpes3`,
`puts3`, `callesid3`, `girid3`, `ssid3`, `golpes4`, `puts4`, `callesid4`,
`girid4`, `ssid4`, `golpes5`, `puts5`, `callesid5`, `girid5`, `ssid5`,
```

```
`golpes6`, `puts6`, `callesid6`, `girid6`, `ssid6`, `golpes7`, `puts7`,
`callesid7`, `girid7`, `ssid7`, `golpes8`, `puts8`, `callesid8`, `girid8`,
`ssid8`, `golpes9`, `puts9`, `callesid9`, `girid9`, `ssid9`, `golpes10`,
`puts10`, `callesid10`, `girid10`, `ssid10`, `golpes11`, `puts11`,
`callesid11`, `girid11`, `ssid11`, `golpes12`, `puts12`, `callesid12`,
`girid12`, `ssid12`, `golpes13`, `puts13`, `callesid13`, `girid13`,
`ssid13`, `golpes14`, `puts14`, `callesid14`, `girid14`, `ssid14`,
`golpes15`, `puts15`, `callesid15`, `girid15`, `ssid15`, `golpes16`,
`puts16`, `callesid16`, `girid16`, `ssid16`, `golpes17`, `puts17`,
`callesid17`, `girid17`, `ssid17`, `golpes18`, `puts18`, `callesid18`,
`girid18`, `ssid18`) VALUES
(1, 2, 2, 1, '2008-06-01 00:00:00', '1', 'Todo perfecto', 6, 2, 1, 1, 1, 5,
3, 1, 1, 1, 5, 2, 1, 1, 1, 4, 2, 1, 1, 1, 5, 2, 1, 1, 1, 6, 3, 1, 1, 1, 5,
3, 1, 1, 1, 4, 2, 1, 1, 1, 7, 2, 1, 1, 1, 4, 2, 1, 1, 1, 4, 3, 1, 1, 1, 7,
3, 1, 1, 1, 6, 2, 1, 1, 1, 3, 2, 1, 1, 1, 5, 2, 1, 1, 1, 8, 4, 1, 1, 1, 4,
1, 1, 1, 1, 3, 1, 1, 1, 1),
(3, 1, 2, 1, '2008-06-01 00:00:00', 'Campeonato', 'Mucho viento', 8, 3, 3,
1, 3, 4, 2, 1, 1, 1, 5, 3, 1, 1, 1, 4, 1, 1, 1, 1, 5, 2, 1, 1, 1, 5, 3, 1,
1, 1, 5, 2, 1, 1, 1, 4, 1, 1, 1, 1, 6, 3, 1, 1, 1, 4, 2, 1, 1, 1, 4, 2, 1,
1, 1, 5, 2, 1, 1, 1, 4, 2, 1, 1, 1, 3, 1, 1, 1, 1, 4, 2, 1, 1, 1, 5, 2, 1,
1, 1, 4, 1, 1, 1, 1, 3, 1, 1, 1, 1),
(4, 1, 1, 1, '2008-06-01 00:00:00', 'Campeonato 2', 'Lluvia', 5, 3, 3, 1,
3, 5, 3, 1, 1, 1, 4, 2, 1, 1, 1, 4, 2, 1, 1, 1, 4, 2, 1, 1, 1, 7, 2, 1, 1,
1, 5, 2, 1, 1, 1, 4, 2, 1, 1, 1, 5, 3, 1, 1, 1, 3, 1, 1, 1, 1, 4, 2, 1, 1,
1, 5, 2, 1, 1, 1, 4, 2, 1, 1, 1, 4, 3, 1, 1, 1, 6, 2, 1, 1, 1, 4, 2, 1, 1,
1, 4, 2, 1, 1, 1, 4, 2, 1, 1, 1),
(5, 2, 1, 2, '2008-06-03 00:00:00', 'Partida con amigos', 'Nada que
comentar', 5, 3, 4, 1, 4, 4, 3, 1, 1, 1, 6, 2, 1, 1, 1, 5, 3, 1, 1, 1, 6,
2, 1, 1, 1, 4, 2, 1, 1, 1, 4, 2, 1, 1, 1, 5, 2, 1, 1, 1, 5, 1, 1, 1, 1, 5,
1, 1, 1, 1, 4, 1, 1, 1, 1, 4, 1, 1, 1, 1, 5, 2, 1, 1, 1, 3, 1, 1, 1, 1, 5,
2, 1, 1, 1, 5, 2, 1, 1, 1, 3, 1, 1, 1, 1, 4, 2, 1, 1, 1),
(6, 1, 1, 1, '2008-06-03 00:00:00', 'Prueba total', 'Nada que comentar', 1,
1, 1, 1, 1, 4, 2, 1, 1, 1, 4, 2, 1, 1, 1, 3, 1, 1, 1, 1, 3, 1, 1, 1, 1, 5,
3, 1, 1, 1, 7, 2, 1, 1, 1, 5, 3, 1, 1, 1, 4, 1, 1, 1, 1, 4, 1, 1, 1, 1, 5,
3, 1, 1, 1, 6, 2, 1, 1, 1, 3, 1, 1, 1, 1, 3, 2, 1, 1, 1, 5, 3, 1, 1, 1, 6,
3, 1, 1, 1, 5, 3, 1, 1, 1, 3, 1, 1, 1, 1),
(7, 1, 1, 1, '2008-06-03 00:00:00', 'Guardo total', 'Viento racheado', 5,
2, 1, 1, 1, 5, 2, 1, 1, 1, 6, 3, 1, 1, 1, 6, 2, 1, 1, 1, 5, 2, 1, 1, 1, 6,
2, 1, 1, 1, 5, 2, 1, 1, 1, 8, 2, 1, 1, 1, 6, 2, 1, 1, 1, 5, 2, 1, 1, 1, 6,
2, 1, 1, 1, 4, 2, 1, 1, 1, 5, 2, 1, 1, 1, 5, 2, 1, 1, 1, 4, 2, 1, 1, 1, 5,
2, 1, 1, 1, 7, 2, 1, 1, 1, 5, 1, 1, 1, 1);
```

Tees

```
INSERT INTO `tees` (`id`, `descripcion`) VALUES
(1, 'Amarillas'),
(2, 'Rojas');
```

Usuarios

```
INSERT INTO `usuarios` (`id`, `user`, `passwd`, `nombre`, `apellidos`,
`fechanac`, `sexo`, `handicap`) VALUES
(1, 'jl', 'jl', 'Administrador', NULL, '2008-05-26 00:00:00', 'H', 24),
(2, 'jlgarcia', 'jlgarcia', 'José Luis', 'García Deza', '2008-05-26
00:00:00', 'H', 36),
```

Anexo C

Instalación del componente.

A continuación se describen las librerías utilizadas para la implementación del componente. Se han utilizado las librerías JRE de MyEclipse 6.0 y J2EE 1.4.

A las que se les han añadido las del framework utilizado Myfaces.

Como el tamaño de las librerías no es significante se han incluido en los entregables del proyecto.

Se acompaña el fichero build.xml que generará la librería del componente y la dejará lista para su uso en la carpeta /dist.

Instalación de la Aplicación de Ejemplo.

Además de las librerías JRE de MyEclipse 6.0 y J2EE 1.4, se han utilizado:

JSF 1.1

http://www.apache.org/dyn/closer.cgi/myfaces/binaries/myfaces-core-1.1.5-bin.zip

hibernate 3.2.6ga

http://sourceforge.net/project/showfiles.php?group_id=40712&package_id=127784&release_id=574498

Spring 2.5.4

http://sourceforge.net/project/showfiles.php?group_id=73357&package_id=173644&release_id=595476

Mysql Connector 3.1

http://dev.mysql.com/downloads/connector/j/3.1.html

En los entregables se han eliminado las librerías hibernate 3.2.6ga y MyFaces 1.1.5, que tendrán que añadirse manualmente, ya que son las que más ocupan. El resto se han dejado para evitar mayores molestias.