



UNIVERSIDADE DA CORUÑA

FACULTADE DE INFORMÁTICA

Departamento de Tecnoloxías da Información e as Comunicaci3ns

PROXECTO DE FIN DE CARREIRA EN
ENXEÑERÍA TÉCNICA INFORMÁTICA DE SISTEMAS

**Diseño e implementación de una
aplicación web J2EE con
arquitectura MVC. Caso de estudio:
sitio web de apuestas deportivas**

Autor: Antonio Fernández Zaragoza

Tutor: Manuel Álvarez Díaz

A Coruña, Junio 2007

Proyecto: Diseño e implementación de una aplicación web J2EE con arquitectura MVC. Caso de estudio: sitio web de apuestas deportivas.

No está permitida la reproducción total o parcial de este proyecto, ni su tratamiento informático, ni la transmisión de ninguna forma o por cualquier medio, ya sea electrónico, mecánico, por fotocopia, por registro u otros métodos, sin el permiso previo y por escrito del autor del mismo.

Dedicatoria

A mis padres y hermano

Antonio Fernández Zaragoza

Diseño e implementación de una aplicación web J2EE con
arquitectura MVC. Caso de estudio: sitio web de apuestas
deportivas

Autor: *Antonio Fernández Zaragoza*

Tutor: *Manuel Álvarez Díaz*

Miembros del Tribunal:

Manuel Álvarez Díaz

Fernando Bellas Permuy

Juan Raposo Santiago

Fecha de lectura:

13 de Julio del 2007

Calificación:

--

AGRADECIMIENTOS

Tengo que dar las gracias a todos aquellos que me han apoyado y ayudado en el desarrollo del proyecto y durante estos cuatro largos años de carrera.

Por una parte, agradecer todo el apoyo que me ha dado siempre mi familia. De forma especial a mis padres, por confiar siempre en mi y darme todo lo que he necesitado a lo largo de todo este tiempo. A mi hermano por siempre estar ahí cuando lo he necesitado y a mis abuelos, por su gran aportación durante los cuatro años de estudios.

Por otro lado, agradezco mucho toda la ayuda que siempre me han ofrecido mis amigos y compañeros, ya que sin ella, todo hubiera sido mucho mas difícil.

RESUMEN

Título: Diseño e implementación de una aplicación Web J2EE con arquitectura MVC. Caso de estudio: sitio web de apuestas deportivas

Autor: Antonio Fernández Zaragoza

Tutor: Manuel Álvarez Díaz

El proyecto aquí documentado consiste en el diseño e implementación de una aplicación web destinada a la realización de apuestas deportivas, ofreciendo al usuario diferentes posibilidades de apuesta sobre variedad de deportes y a los administradores una fácil gestión de dichas apuestas.

El usuario podrá moverse fácilmente por el árbol de categorías, visualizando los eventos presentes en cada deporte y los tipos de apuesta que se le ofrece en cada evento. Una vez seleccionadas las opciones por las cuales optará el usuario, podrá gestionarlas en un talón de apuestas, pudiendo eliminarlas, añadir más opciones o finalizar cada una de las apuestas, con la posibilidad de escoger entre apuesta simple o combinada si es posible. En todo momento, el usuario podrá consultar información sobre sus apuestas así como los movimientos económicos de su cuenta. Comentar que la aplicación ofrece internacionalización. Por otra parte están los administradores, encargados de manipular y gestionar las categorías, eventos y demás elementos que componen los servicios ofrecidos al usuario.

La aplicación está dotada de gran escalabilidad, con la posibilidad de agregar en un futuro nuevas funcionalidades sin necesidad de grandes cambios en la aplicación.

Palabras Clave:

Maven2, Aplicación web, Java, J2EE, JDBC, JSP, JSTL, J2SE, MySQL, Servlet, Struts, Tiles, Tomcat, XML, JUnit, apuestas, patrones

ÍNDICE DE CONTENIDOS

1. INTRODUCCIÓN	2
1.1. DETERMINACIÓN DE LA SITUACIÓN ACTUAL	2
1.2. ALCANCE Y OBJETIVOS	3
2. HERRAMIENTAS DISPONIBLES	5
2.1. APLICACIONES EMPRESARIALES	5
2.2. PROCESO UNIFICADO DE DESARROLLO SOFTWARE	6
2.3. LENGUAJE UNIFICADO DE MODELADO (UML)	8
2.4. J2EE	9
2.5. JDBC	9
2.6. MYSQL	9
2.7. APACHE TOMCAT	10
2.8. ECLIPSE	10
2.9. MAVEN 2	10
2.10. JSTL	11
2.11. JNDI	11
2.12. JSP	12
2.13. XML	12
2.14. XHTML	13
2.15. CSS	13
2.16. STRUTS	13
3. ESTUDIO DE VIABILIDAD	15
3.1. COSTES	15
3.2. RECURSOS	15
3.3. PERSONAL	15
4. INTRODUCCIÓN AL DESARROLLO REALIZADO	16
4.1. ITERACIONES	17
5. PLANIFICACIÓN Y EVALUACIÓN DE COSTES	19
5.1. PLANIFICACIÓN INICIAL REALIZADA	19
5.2. EVALUACIÓN DE LA PLANIFICACIÓN REALIZADA	22
6. REQUISITOS DEL SISTEMA	23
6.1. INTRODUCCIÓN	23
6.2. ACTORES	26
6.3. CASOS DE USO	27

6.4.	MODELO DE CASOS DE USO	30
7.	DISEÑO DE LA APLICACIÓN	32
7.1.	ARQUITECTURA GENERAL	34
7.2.	SUBSISTEMA 1	35
7.2.1.	Objetivos	35
7.2.2.	Arquitectura	35
7.2.3.	Modelo	37
7.2.3.1.	Clases Persistentes	37
7.2.3.2.	Interfaces de las Fachadas	44
7.2.4.	Controlador	56
7.2.5.	Vista	60
7.3.	SUBSISTEMA 2	62
7.3.1.	Objetivos	62
7.3.2.	Arquitectura	62
7.3.3.	Modelo	63
7.3.4.	Controlador	63
8.	IMPLEMENTACIÓN	65
8.1.	SOFTWARE REQUERIDO	65
8.2.	ESTRUCTURA	65
8.3.	INSTRUCCIONES DE COMPILACIÓN	66
9.	PRUEBAS	68
10.	CONCLUSIONES Y FUTURAS LÍNEAS DE TRABAJO	71
11.	APÉNDICE	73
11.1.	INSTALACIÓN DEL SOFTWARE / MANUAL DE USUARIO	73
11.2.	CONTENIDO DEL CD	79
12.	BIBLIOGRAFÍA	81
12.1.	ENLACES DE INTERÉS	83
13.	GLOSARIO	84
14.	ACRÓNIMOS	86

ÍNDICE DE FIGURAS

1 EJEMPLO DE APLICACIÓN EMPRESARIAL CON UNA ARQUITECTURA EN TRES CAPAS	6
2 PROCESO UNIFICADO. ÉNFASIS DE CADA DISCIPLINA A LO LARGO DE LAS DIFERENTES FASES	7
3. DIFERENTES CAPAS DE LA APLICACIÓN JUNTO CON ALGUNAS TECNOLOGÍAS USADAS	17
4 DESGLOSE DE LAS TAREAS PLANIFICADAS POR FECHAS Y SU DURACIÓN EN DÍAS	20
5 DIAGRAMA DE GANTT DE LA PLANIFICACIÓN REALIZADA.....	21
6. DIAGRAMA DE CASOS DE USO DEL ADMINISTRADOR	30
7. DIAGRAMA DE CASOS DE USO DEL USUARIO REGISTRADO Y AUTENTICADO	31
8. DIAGRAMA DE LOS CASOS DE USO GENERALES PARA CUALQUIER USUARIO	31
9. DIAGRAMA DEL PATRÓN DAO	33
10. DIAGRAMA DE CLASES DE APACHE STRUTS	57
11. DIAGRAMA DE SECUENCIA DEL PROCESO QUE SIGUE UNA PETICIÓN	59
12. DIAGRAMA DE CLASE DE LA FACHADA SESSIONMANAGER	60
13. GENERACIÓN DE UNA VISTA A PARTIR DE UNA PETICIÓN.....	61
14. ARQUITECTURA EN PAQUETES DEL SUBSISTEMA 2	63
15. DIAGRAMA DE CLASES DEL PAQUETE DE EXCEPCIONES DEL SUBSISTEMA 2	64
16. ESTRUCTURA DE LA APLICACIÓN PARA SU COMPILACIÓN CON MAVEN 2	65
17. ESTRUCTURA EN PAQUETES DE LAS PRUEBAS DE UNIDAD	68
18. PÁGINA PRINCIPAL DE APUESTASINPARAR.ES.....	76
19. MENÚ DE LA CUENTA DE USUARIO DE APUESTASINPARAR.ES.....	76
20. CREACIÓN DE UN EVENTO ESPECÍFICO DE FÚTBOL POR PARTE DEL ADMINISTRADOR	77
21. CREACIÓN DE UN TIPO DE APUESTA POR PARTE DEL ADMINISTRADOR.....	78
22. USUARIO INTRODUCIENDO OPCIONES DE APUESTA EN SU TALÓN DE APUESTAS.....	78
23. USUARIO ADMINISTRANDO SU TALÓN DE APUESTAS	79

NOTA: Se adjunta un disco con el sistema propuesto, documentación y una aplicación de prueba.

1. Introducción

1.1. Determinación de la situación actual

El negocio de las apuestas ha encontrado en Internet un socio ideal, ya que en muchos países las loterías y apuestas son ilegales o están en manos del Estado, la única forma de acceder a ellas es a través de Internet, en casas de apuestas radicadas en países que legalmente permiten los juegos de azar. Las casas de apuestas en Internet facilitan a sus usuarios un sistema sencillo y divertido para apostar en los eventos deportivos de mayor trascendencia, fútbol, baloncesto, tenis, motociclismo, Fórmula 1, carreras de caballos y cualquier deporte imaginable. De esta forma los jugadores pueden apostar en el deporte que mejor conocen y obtener suculentos beneficios, además el riesgo de perder y la oportunidad de ganar consiguen generar enormes dosis de adrenalina que animan al usuario a participar.

Actualmente ya existe gran variedad de páginas de apuestas deportivas o casas de apuestas online, como por ejemplo BetAndWin, Ladbrokes y Miapuesta.com. Todas ellas ofrecen la posibilidad de realizar gran variedad de apuestas deportivas de forma sencilla, fiable y entretenida.

No cabe duda de que el éxito de este tipo de páginas es total, siendo un gran negocio para las casas de apuestas. Cada vez se ganan más clientes, principalmente por la sencillez a la hora de apostar y la fiabilidad que ofrecen a los usuarios, dando confianza a muchos internautas que en un principio podían ser reacios a realizar estas tareas vía Internet. Pero no es simplemente apostar, si no que el usuario tiene acceso a variedad de información, alguna imprescindible, como el historial de apuestas o el histórico de movimientos monetarios de sus cuentas, y otras no menos interesantes, como información deportiva sobre los eventos deportivos, ayudando a la hora de apostar.

1.2. Alcance y Objetivos

El objetivo de esta aplicación web de apuestas deportivas es ofrecer a cualquier usuario de la red (mayor de edad) la posibilidad de realizar gran variedad de apuestas deportivas, sin moverse de casa, con gran facilidad y de forma segura.

La oferta de tipos de apuestas es muy amplia, ofreciendo desde deportes populares como el fútbol o baloncesto a otros quizá desconocidos para muchos, como carrera de galgos. Inicialmente solo habrá información deportiva de la primera división de liga española, pero la aplicación está preparada para la incorporación de información deportiva de cualquier deporte, pudiendo ser consultada en cualquier momento por el usuario ya sea por el menú de información deportiva o pinchando en un enlace que se ofrecerá en aquellas apuestas donde sea posible ofrecer información deportiva. En cuanto al sistema de apuesta, la sencillez es su gran cualidad, el usuario podrá recorrer todo los deportes junto a sus categorías, subcategorías y eventos, cuando encuentre un tipo de apuesta que le interese, seleccionará una opción que se incorpora a su “talón de apuestas personal”, y seguir eligiendo apuestas. Cuando el usuario se decida a apostar finalmente, accede al talón, donde aparecen todas las apuestas que previamente introdujo, todas ellas con todos sus detalles. El talón ofrece al usuario una visión clara y rápida de las apuestas que quiera realizar, permitiéndole una completa gestión con operaciones tales como eliminar una opción de apuesta que no le interese o refrescar el talón en busca de posibles cambios en los tipos de apuesta.

Entre las opciones ofrecidas, se destaca la posibilidad de consultar de forma transparente el historial de apuestas y todos los movimientos monetarios de su cuenta, ya que el usuario tiene en todo momento la información que pueda necesitar, dándole seguridad y confianza. Ambas búsquedas se pueden realizar indicando una fecha de inicio y otra de fin, o si se prefiere un periodo de tiempo. En cuanto a las apuestas se puede hacer un filtrado según su estado (pendiente, ganada, perdida o cancelada) y para los movimientos bancarios se podrá diferenciar por el tipo de movimiento (ingreso, retiro, ingreso por apuesta y retiro por apuesta). Por último, el cliente podrá registrar diferentes tipos de tarjetas bancarias para el ingreso de saldo a través de ellas, y cuando quiera retirar sus ganancias a una cuenta bancaria, simplemente tendrá que rellenar un formulario y decidir cuando dinero quiere retirar.

En cuanto a la administración de la aplicación, los usuarios administradores navegan por el árbol de categorías, pudiendo en cualquier momento añadir o eliminar categorías y eventos, crear tipos de apuestas junto con sus opciones de apuestas y su posterior control. En cuanto a la creación de eventos, en el caso de partidos de fútbol de primera división de la liga española, tendrá un “Wizard”, donde ya podrá elegir los dos equipos enfrentados, creando así un vínculo entre ese evento y la información deportiva de esos dos equipos. En un futuro, se podrán crear Wizards para diferentes deportes, sin apenas realizar cambios en nuestra aplicación.

2. Herramientas Disponibles

En este capítulo se describen las herramientas y tecnologías utilizadas en el desarrollo la aplicación, así como conceptos directamente implicados en el proyecto. Se comienza con la idea de aplicación empresarial.

2.1. Aplicaciones empresariales

Una aplicación web empresarial debe cumplir las siguientes características

- **Escalabilidad**

Ha de ofrecer una buena escalabilidad tanto horizontal como vertical de modo que si aumenta la carga del sistema podamos añadir servidores o ampliar los existentes sin que sea necesario realizar modificaciones.

- **Mantenibilidad**

Ha de permitir añadir modificar los componentes existentes sin que se modifique el comportamiento del sistema.

- **Fiabilidad**

- **Disponibilidad**

Hemos de tener el soporte de arquitecturas tolerantes a fallos, sistemas de redundancia, etc., que nos aseguren que el sistema estará siempre disponible.

- **Extensibilidad**

Ha de ser posible añadir nuevos componentes y capacidades al sistema sin que se vean afectados el resto de componentes.

- **Manejabilidad**

Nuestros sistemas han de ser fácilmente manejables y configurables.

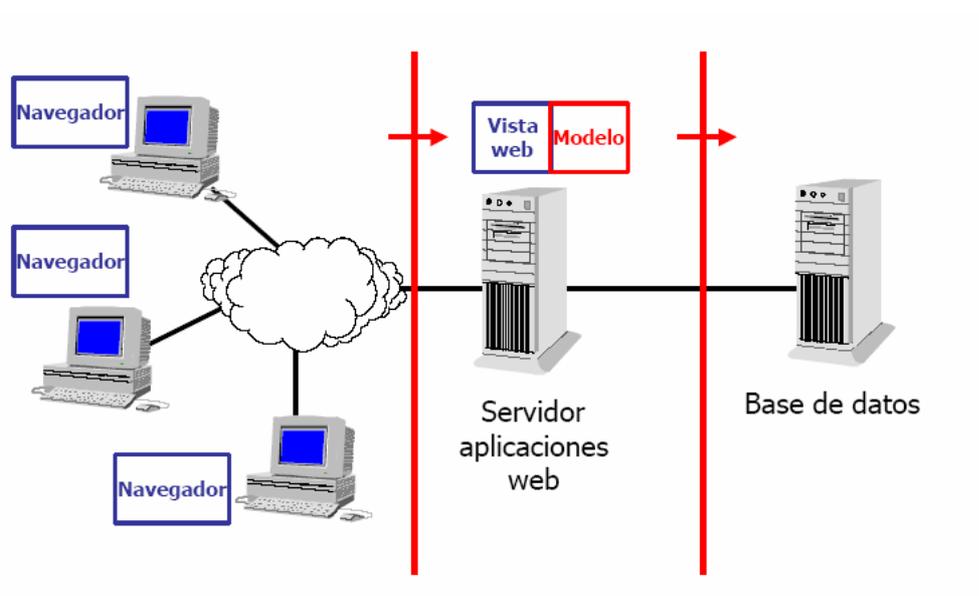
- **Seguridad**

Hemos de tener buenos sistemas de seguridad tanto a nivel de autenticación, como de autorización y como de transporte.

- **Rendimiento**

Se ha de ofrecer automáticamente soporte de clustering, balanceo de carga, pools de objetos, pools de conexiones, cachés, y en general mecanismos que permitan aumentar el rendimiento de manera transparente al usuario.

En cuanto a la arquitectura, una buena opción es la propuesta en tres capas frente a otras alternativas como puede ser una modelo en cuatro capas. Aunque físicamente las capas vista y modelo se encuentren en la misma máquina, lógicamente están bien diferenciadas.



1 Ejemplo de aplicación empresarial con una arquitectura en tres capas

2.2. Proceso Unificado de Desarrollo Software

El Proceso Unificado de Desarrollo Software o simplemente Proceso Unificado es un marco de desarrollo software iterativo e incremental. El refinamiento más conocido y documentado del Proceso Unificado es el Proceso Unificado de Rational o simplemente RUP. Sus características son las siguientes.

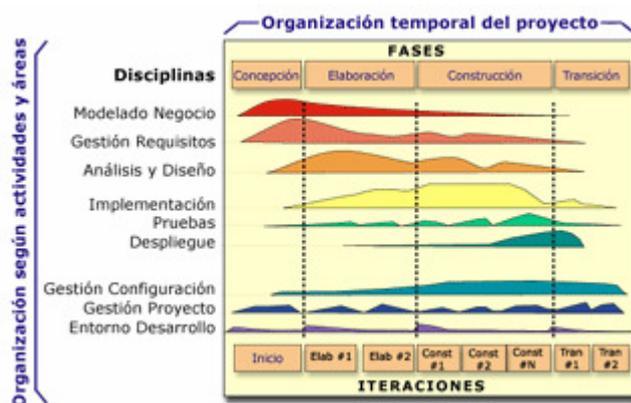
• Iterativo e Incremental

El desarrollo es iterativo e incremental, compuesto de cuatro fases denominadas Inicio, Elaboración, Construcción y Transición. Cada una de estas fases es a su vez dividida en una serie de iteraciones (la de inicio sólo consta de varias iteraciones en proyectos grandes). Estas iteraciones ofrecen como resultado un incremento del producto desarrollado que añade o mejora las funcionalidades del sistema en desarrollo.

Cada una de estas iteraciones se divide a su vez en una serie de disciplinas que recuerdan a las definidas en el ciclo de vida clásico o en cascada: Análisis de requisitos, Diseño, Implementación y Prueba. Aunque todas las iteraciones suelen incluir trabajo en casi todas las disciplinas, el grado de esfuerzo dentro de cada una de ellas varía a lo largo del proyecto.

• Dirigido por los casos de uso

En el Proceso Unificado los casos de uso se utilizan para capturar los requisitos funcionales y para definir los contenidos de las iteraciones. La idea es que cada iteración tome un conjunto de casos de uso o escenarios y desarrolle todo el camino a través de las distintas disciplinas: diseño, implementación, prueba, etc. el proceso dirigido por casos de uso es el rup.



2 Proceso Unificado. Énfasis de cada disciplina a lo largo de las diferentes fases

• Centrado en la arquitectura

El Proceso Unificado asume que no existe un modelo único que cubra todos los aspectos del sistema. Por dicho motivo existen múltiples modelos y vistas que definen la

arquitectura de software de un sistema. La analogía con la construcción es clara, cuando construyes un edificio existen diversos planos que incluyen los distintos servicios del mismo: electricidad, fontanería, etc.

- **Enfocado en los riesgos**

El Proceso Unificado requiere que el equipo del proyecto se centre en identificar los riesgos críticos en una etapa temprana del ciclo de vida. Los resultados de cada iteración, en especial los de la fase de Elaboración, deben ser seleccionados en un orden que asegure que los riesgos principales son considerados primero.

El Proceso Unificado no es simplemente un proceso, sino un marco de trabajo extensible que puede ser adaptado a organizaciones o proyectos específicos. De la misma forma, el *Proceso Unificado de Rational*, también es un marco de trabajo extensible, por lo que muchas veces resulta imposible decir si un refinamiento particular del proceso ha sido derivado del Proceso Unificado o del RUP. Por dicho motivo, los dos nombres suelen utilizarse para referirse a un mismo concepto.

2.3. Lenguaje Unificado de Modelado (UML)

Lenguaje Unificado de Modelado (UML), es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

Es importante remarcar que UML es un "lenguaje" para especificar y no un método o un proceso, se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar y construir.

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas.

2.4. J2EE

Java 2 Enterprise Edition [2][8] es un conjunto de especificaciones de APIs Java distribuidas por Sun para el desarrollo de aplicaciones empresariales, está basado en J2SE y la mayor parte de las abstracciones de las APIs se corresponden con interfaces y clases abstractas. J2EE proporciona funcionalidades para aplicaciones empresariales tales como el acceso de base de datos (JDBC), utilización de directorios distribuidos (JNDI), funciones de correo electrónico (JavaMail), acceso a métodos remotos (RMI/CORBA) y aplicaciones Web (Servlet y JSP).

J2EE es una especificación, de forma que existen implementaciones de diferentes fabricantes como por ejemplo IBM WebSphere y Oracle Application Server, incluso OpenSource como Tomcat.

Existen alternativas a J2EE tales como .NET de Microsoft o LAMP. He elegido J2EE para el desarrollo del proyecto entre otros por su madurez, su versatilidad y que hace posible crear arquitecturas completas basadas única y exclusivamente en productos de software libre.

2.5. JDBC

JDBC (Java Database Connectivity) es un API que permite lanzar queries sobre bases de datos relacionales desde el lenguaje de programación Java independientemente del la base de datos a la cual se accede utilizando el lenguaje SQL del modelo de base de datos que se utilice. Actualmente forma parte de J2SE.

Para poder operar con la base de datos es necesario un driver adecuado para ella. Este driver suele implementar todos las interfaces del API de JDBC. Nuestro código nunca depende del driver y siempre trabajará contra los paquetes `java.sql` y `javax.sql`.

2.6. MySQL

MySQL es un sistema gestor de bases de datos relacionales con funciones avanzadas de administración y optimización de bases de datos para facilitar las tareas habituales. Implementa funcionalidades Web, permitiendo una acceso seguro y sencillo a los datos a través de Internet. Este sistema gestor de base de datos incluye capacidades de análisis integradas, servicios de transformación y duplicación de datos y funciones de programación

mejoradas. Es OpenSource y se puede decir que es un sistema cliente servidor de administración de bases de datos relacionales diseñado para el trabajo tanto en los sistemas operativos Windows como en sistemas UNIX/LINUX. Como alternativa se puede hablar de otros gestores comerciales como Oracle, SQLServer o también OpenSource como PostgreSQL o FireBird.

En la aplicación he utilizado MySQL debido a que buscaba un gestor de base de datos relacional ampliamente utilizado, OpenSource y con gran madurez.

2.7. Apache Tomcat

Tomcat [3] consiste en un servidor (contenedor) OpenSource de aplicaciones web J2EE con soporte para Servlet 2.4 y JSP 2.0. Contiene el compilador Jasper, que compila las JSPs convirtiéndolas en servlets. Se puede decir que es multiplataforma ya que está implementado en Java. Hoy en día goza de gran popularidad y se considera óptimo como servidor autónomo en entornos con alto nivel de tráfico y alta disponibilidad. Estas características, junto a que es OpenSource y no se va a hacer uso de EJB, hicieron que eligiera Apache Tomcat como contenedor de mi aplicación web.

2.8. ECLIPSE

Eclipse es un IDE multiplataforma OpenSource para crear aplicaciones clientes de cualquier tipo. Una de sus grandes ventajas es que basa su funcionamiento en plugins con lo que es ampliable para que haga prácticamente cualquier cosa, desde edición de XML hasta el control de Tomcat, pasando por plugins para el resaltado de sintaxis. Posee un potente editor de texto con resaltado de sintaxis, compila a tiempo real, permite implementar pruebas de unidad con JUnit y pensando en el caso de este proyecto, se puede integrar Maven 2.

2.9. MAVEN 2

Herramienta software [1] para la gestión del ciclo de vida de un proyecto software. Puede ser visto como un framework o un entorno de trabajo para la gestión de proyectos y despliegue de aplicaciones. Consta de un núcleo, POM (Project Object Model), que ejecuta distintas acciones o goals y contiene una descripción detallada del proyecto, incluyendo información de versiones, gestión de configuración, dependencias, recursos de la aplicación

y de pruebas, miembros del equipo...etc. Dichos goals se encuentran distribuidos en plugins. La versatilidad de Maven se encuentra en la cantidad ingente de plugins que tiene disponible. Maven 2 compila un proyecto Java, ejecuta test unitarios de JUnit, genera paquetes jar, wars, ears o distribuciones zip, además de generar reports. Se guía por un fichero de configuración pom.xml en el directorio raíz dónde se definen los recursos de los que depende el proyecto.

Maven 2 es una herramienta de más alto nivel que Ant, y a diferencia de éste, trata de forma automática las dependencias del proyecto y se actualiza en línea mediante servidores repositorios. Es capaz de descargar nuevas actualizaciones de las bibliotecas de las que depende el proyecto y de igual manera subir una nueva distribución a un repositorio de versiones, dejándola al acceso de todos los usuarios del proyecto Proporciona una gestión transparente de códigos y librerías entre los distintos desarrolladores del proyecto.

Me he decantado por Maven 2 ya que como se ha comentado, presenta bastantes ventajas frente a otra opción como podría ser Ant, siendo más potente, extensible y estando más orientado a la gestión total del proyecto a lo largo de su ciclo de vida.

2.10. JSTL

JSP Standard Tag Library [13] es una librería general de tags estándar. Esta librería permite el control de flujo, tiene soporte para internacionalización, capacidad de parsear y recorrer un documento XML, acceso a base de datos y además definir un conjunto de funciones estándar. Gracias a los tags, se podrá realizar unas paginas JSP limpias de código java, permitiendo la separación de los roles de programador y diseñador gráfico de páginas web, ya que estos últimos solo necesitarían tener conocimiento de los tags.

2.11. JNDI

JNDI (Interfaz de Nombrado y Directorio Java) es una Interfaz de Programación de Aplicaciones para servicios de directorio. Esto permite a los clientes descubrir y buscar objetos y nombres a través de un nombre y, como todas las APIs de Java que hacen de interfaz con sistemas host, es independiente de la implementación subyacente. Adicionalmente, especifica una interfaz de proveedor de servicio (SPI) que permite que las

implementaciones del servicio de directorio sean enchufadas en el framework. Las implementaciones pueden hacer uso de un servidor, un fichero, o una base de datos.

2.12. JSP

JavaServer Pages (JSP) [7] es una tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo.

Las JSP's permiten la utilización de código Java mediante scripts. Además es posible utilizar algunas acciones JSP predefinidas mediante etiquetas. Estas etiquetas pueden ser enriquecidas mediante la utilización de Librerías de Etiquetas (TagLibs o Tag Libraries) externas e incluso personalizadas.

Microsoft, la más directa competencia de Sun, ha visto en esta estrategia de Sun una amenaza, lo que le ha llevado a que su plataforma .NET incluya su lenguaje de scripts ASP.NET que permite ser integrado con clases .NET (ya estén hechas en C++, VisualBasic o C#) del mismo modo que jsp se integra con clases Java.

2.13. XML

XML (eXtensible Markup Language) es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

2.14. XHTML

XHTML [10], acrónimo inglés de eXtensible Hypertext Markup Language (lenguaje extensible de marcado de hipertexto), es el lenguaje de marcado pensado para sustituir a HTML como estándar para las páginas web. XHTML es la versión XML de HTML, por lo que tiene, básicamente, las mismas funcionalidades, pero cumple las especificaciones, más estrictas, de XML. Su objetivo es avanzar en el proyecto del World Wide Web Consortium para lograr una web semántica, donde la información, y la forma de presentarla estén claramente separadas. En este sentido, XHTML serviría únicamente para transmitir la información que contiene un documento, dejando para hojas de estilo (como las hojas de estilo en cascada) y JavaScript su aspecto y diseño en distintos medios (ordenadores, PDAs, teléfonos móviles, impresoras...).

2.15. CSS

Hojas de Estilo en Cascada (Cascading Style Sheets) [14], es un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir, o incluso cómo va a ser pronunciada la información presente en ese documento a través de un dispositivo de lectura. Esta forma de descripción de estilos ofrece a los desarrolladores el control total sobre estilo y formato de sus documentos.

2.16. Struts

Struts [6] [11] es un framework OpenSource de soporte para el desarrollo de aplicaciones Web siguiendo el patrón MVC bajo la plataforma J2EE (Java 2, Enterprise Edition). Funciona sobre cualquier servidor de aplicaciones que implemente las APIs de servlets y JSP. Struts se desarrollaba como parte del proyecto Jakarta de la Apache Software Foundation, pero actualmente es un proyecto independiente conocido como Apache Struts.

Struts ofrece soporte para la implementación de las capas controlador y vista de nuestra aplicación proporcionando el Servlet Front Controller y clases relacionadas, según el patrón Front Controller, validación de parámetros, un sistema de plantillas y una completa librería de tags JSP.

Struts permite reducir el tiempo de desarrollo. Su carácter de "software libre" y su compatibilidad con todas las plataformas en que Java Enterprise esté disponible, lo convierte en una herramienta altamente eficiente.

3. Estudio de Viabilidad

A continuación se analizan los factores que determinan que el proyecto es viable.

3.1. Costes

Se analiza una estimación de gastos para observar la rentabilidad del proyecto. En este caso, no se esperan costes excesivos pensados para el tiempo de duración del desarrollo del proyecto contando que no se produzcan excesivos retrasos que puedan suponer costes adicionales. Se estima una rentabilidad del proyecto a corto plazo una vez puesto en funcionamiento.

3.2. Recursos

Tanto a nivel de hardware y software disponemos de todos los recursos necesarios para el correcto desarrollo de la aplicación.

3.3. Personal

Se cuenta con dos empleados, por un lado un Jefe de proyecto y por otro un Analista-Programador, suficientes para desarrollar una aplicación de estas características en el tiempo previsto

4. Introducción al Desarrollo Realizado

Para el desarrollo del proyecto se han utilizado múltiples y variadas tecnologías aplicadas a las diferentes partes o capas en que se divide una aplicación web J2EE. A continuación se comenta un poco el uso de las tecnologías descritas en capítulos anteriores aplicadas a nuestro caso.

Vista

Para la creación de las páginas JSP se ha utilizado XHTML, proporcionando mayor separación entre la información que se quiere mostrar y la presentación en si. A su vez se ha evitado el uso de scriptlets, utilizando en su lugar los tags estándar JSTL y los tags proporcionados por Struts. De esta forma se consigue la separación de roles, con lo que un diseñador web no necesita tener conocimientos de programación en java. En cuanto a la presentación visual, los estilos CSS se encargan de ello. Con la combinación de estas tres tecnologías, se crean páginas JSP limpias y claras.

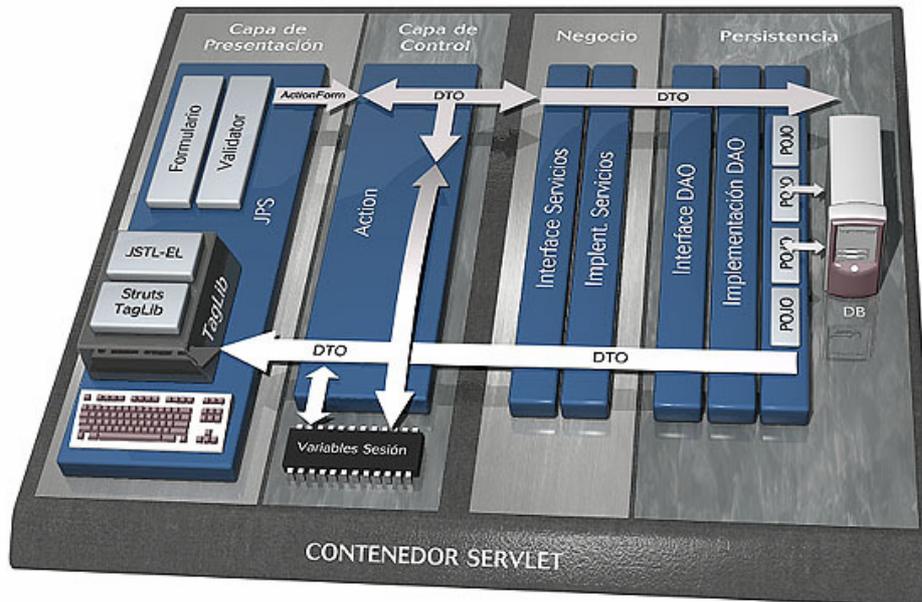
El framework Struts también participa en la capa vista ofreciendo los “Tiles” o sistema de plantillas.

Modelo

El API JDBC nos proporciona las funcionalidades necesarias para que el modelo pueda realizar las transacciones con el repositorio de datos utilizado, en este caso una base de datos relacional. Por otro lado, los datos de configuración requeridos en distintos ámbitos del modelo podrán ser encontrados a través del API JNDI que nos facilita J2EE.

Controlador

Para la capa controlador, se hará uso del framework Struts que implementa el patrón Front Controller y nos proporciona funcionalidades para recoger y validar los datos de los formularios.



3. Diferentes capas de la aplicación junto con algunas tecnologías usadas

Para la creación de la aplicación se ha seguido una metodología de desarrollo basada en iteraciones similar a la descrita en Proceso Unificado. Se plantean que funcionalidades se van a implementar en cada iteración, luego se realizarán las tareas de análisis, diseño, implementación, pruebas y documentación para cada una de estas iteraciones, es decir, un ciclo de vida tradicional. La primera iteración debe proporcionar un prototipo funcional o línea base a la cual se la irá incorporando funcionalidades en cada iteración hasta la última de estas, donde ya se tendrá el producto final requerido. Por otro lado, para la documentación y especificación del proyecto se ha utilizado UML (Unified Modeling Language) y en cuanto a la codificación se ha intentado seguir en la medida de lo posible el estándar de codificación Java.

4.1. Iteraciones

Antes de comenzar el proyecto, se han determinado el número de iteraciones en base al tiempo dispuesto para realizar la aplicación y las funcionalidades que deben implementar. En este caso, se decide realizar dos iteraciones.

- **Iteración 1**

Como se comentó antes, es la iteración más complicada que nos debe ofrecer la base para el resto de las iteraciones. Para ello se seleccionaron las funcionalidades que mejor podían ayudar a dar soporte a las demás. Se implementan las funciones de registro y autenticación de usuario, así como muchos de los casos de uso para el administrador, ya que muchas funcionalidades dependen de la creación de datos, especialmente categorías, eventos y tipos de apuesta. Por otro lado se crea una interfaz gráfica sencilla para interactuar con la aplicación.

- **Iteración 2**

Se implementan el resto de funcionalidades. Para el usuario se crean todos los casos de uso como registrar tarjetas, realizar ingresos, visión de históricos y otros. Para el administrador se añaden mas funciones de control sobre los tipos de apuestas

5. Planificación y Evaluación de Costes

La planificación de un proyecto pretende establecer unos criterios de trabajo con los que poder realizar buenas estimaciones de recursos, coste y plazos temporales. A continuación explica la planificación realizada para este proyecto y se analizan los posibles fallos cometidos.

5.1. Planificación inicial realizada

Se cuenta con dos personas para el desarrollo del proyecto, en primer lugar un ingeniero técnico informático que desempeñará los roles de analista y programador. Por otro lado, un ingeniero informático que trabajará como Jefe de proyecto. Ambas personas se reunirán cada semana para realizar el seguimiento del proyecto.

Para el analista y programador se establece un horario de trabajo diario de lunes a viernes de 10.00 a 14.00 horas por la mañana y de 16.00 a 19.00 por la tarde, con un total de 7 horas diarias. Por otro lado, el horario para el jefe de proyecto será de dos horas semanales para realizar el seguimiento. Se establece un coste por hora de trabajo para cada uno de los empleados de 15 euros la hora. Como se puede observar, la mayor parte del proyecto será llevado a cabo por un solo empleado, por lo que el solapamiento de tareas en el tiempo será imposible, pudiendo a lo sumo intercalar algunas. En ningún caso se solapan iteraciones.

El 12/12/07 se comienza una primera fase que he denotado por Fase de Planificación, donde ambos empleados encargadas del proyecto definen y concretan la metodología de desarrollo y realizan el aprendizaje de las nuevas tecnologías a utilizar.

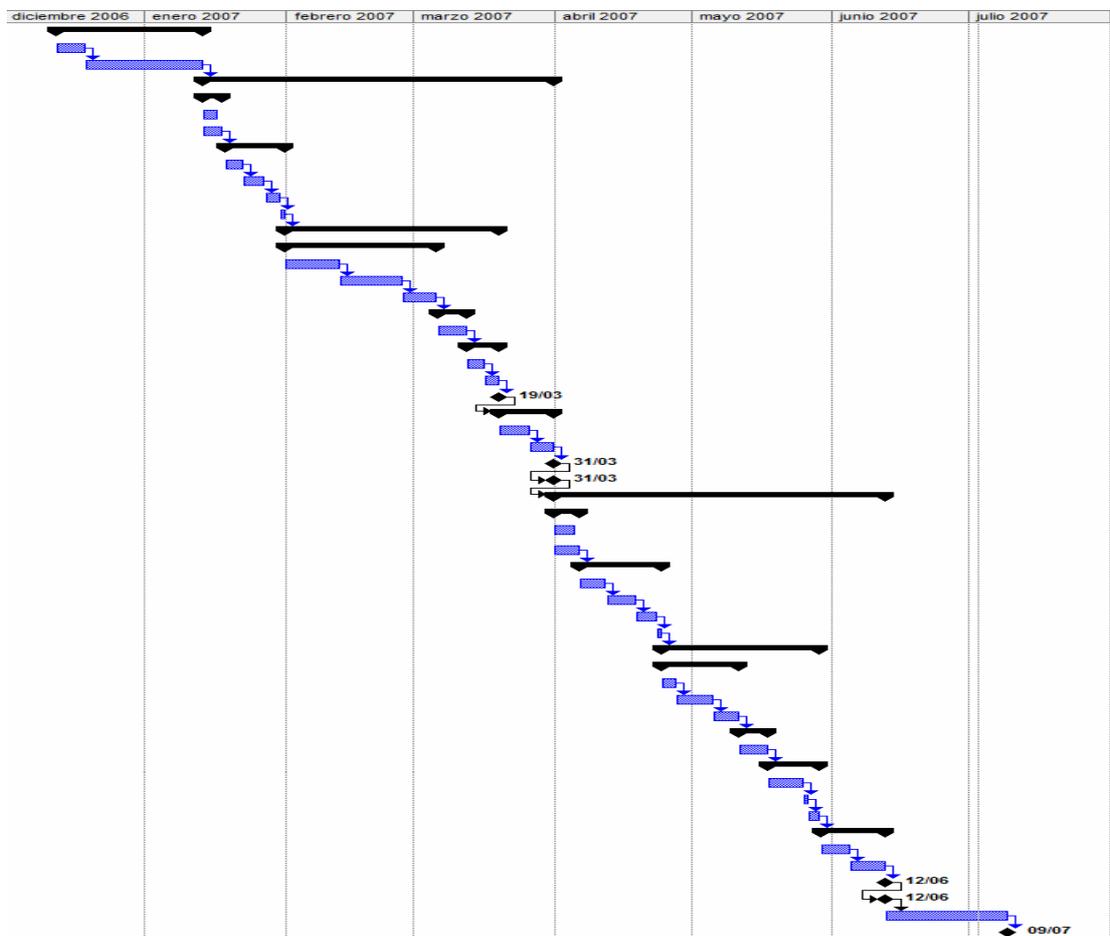
Se establece la fecha de inicio de la primera iteración para el 13/01/2007 y como fecha de finalización del proyecto el 9/07/2007. En la imagen se aprecia la planificación inicial realizada, desglosada en las diferentes iteraciones realizadas y cada uno de los ciclos de desarrollo dentro de cada iteración.

	Nombre de tarea	Duración	Comienzo	Fin	Pred
1	Fase de planificación	25 días	mar 12/12/06	sáb 13/01/07	
2	Definición de metodología a desarrollar	5 días	mar 12/12/06	lun 18/12/06	
3	Definición y aprendizaje de herramientas y tecnologías a utilizar	20 días	lun 18/12/06	sáb 13/01/07	2
4	Iteración numero uno	58 días	sáb 13/01/07	sáb 31/03/07	3
5	Análisis	3 días	sáb 13/01/07	mié 17/01/07	
6	Lectura y análisis de requisitos	2 días	sáb 13/01/07	mar 16/01/07	
7	Selección de subconjunto de casos de uso para la creación de un prototipo funcional	3 días	sáb 13/01/07	mié 17/01/07	
8	Diseño	10 días	jue 18/01/07	mié 31/01/07	7
9	Definición de casos de uso	3 días	jue 18/01/07	lun 22/01/07	
10	Definición de objetos del dominio	4 días	lun 22/01/07	vie 26/01/07	9
11	Definición de fachada junto con sus casos de uso	2 días	sáb 27/01/07	mar 30/01/07	10
12	Reunión seguimiento proyecto	1 día	mar 30/01/07	mié 31/01/07	11
13	Implementación	35 días	mié 31/01/07	lun 19/03/07	12
14	Implementacion de la capa model	25 días	mié 31/01/07	lun 05/03/07	
15	Implementación de VOS	9 días	mié 31/01/07	lun 12/02/07	
16	Implementación de DAOS	10 días	lun 12/02/07	lun 26/02/07	15
17	Implementación de fachadas	6 días	lun 26/02/07	lun 05/03/07	16
18	Implementacion de capa controlador	5 días	mar 06/03/07	lun 12/03/07	17
19	Implementación de las acciones	5 días	mar 06/03/07	lun 12/03/07	
20	Implementación de la vista	5 días	lun 12/03/07	lun 19/03/07	19
21	Creación de las paginas jsp	3 días	lun 12/03/07	vie 16/03/07	
22	Configuración de struts y otras configuraciones	2 días	vie 16/03/07	lun 19/03/07	21
23	Reunión seguimiento del proyecto	0 días	lun 19/03/07	lun 19/03/07	22
24	Pruebas	10 días	lun 19/03/07	sáb 31/03/07	23
25	Pruebas individuales	5 días	lun 19/03/07	lun 26/03/07	
26	Pruebas de unidad	5 días	lun 26/03/07	sáb 31/03/07	25
27	Fin primera de primera iteración	0 días	sáb 31/03/07	sáb 31/03/07	26
28	Reunión seguimiento del proyecto	0 días	sáb 31/03/07	sáb 31/03/07	27
29	Iteración numero dos	54 días	sáb 31/03/07	mar 12/06/07	28
30	Análisis	4 días	sáb 31/03/07	vie 06/04/07	
31	Lectura y análisis de requisitos	3 días	sáb 31/03/07	jue 05/04/07	
30	Análisis	4 días	sáb 31/03/07	vie 06/04/07	
31	Lectura y análisis de requisitos	3 días	sáb 31/03/07	jue 05/04/07	
32	Selección de casos de uso restantes para completar la aplicación	4 días	sáb 31/03/07	vie 06/04/07	
33	Diseño	13 días	vie 06/04/07	mar 24/04/07	32
34	Definición de casos de uso	4 días	vie 06/04/07	mié 11/04/07	
35	Definición de objetos del dominio	5 días	jue 12/04/07	mié 18/04/07	34
36	Definición de fachada junto con sus casos de uso	3 días	mié 18/04/07	lun 23/04/07	35
37	Reunión seguimiento proyecto	1 día	lun 23/04/07	mar 24/04/07	36
38	Implementación	26 días	mar 24/04/07	lun 28/05/07	37
39	Implementacion de la capa model	13 días	mar 24/04/07	vie 11/05/07	
40	Implementación de VOS	3 días	mar 24/04/07	vie 27/04/07	
41	Implementación de DAOS	6 días	vie 27/04/07	sáb 05/05/07	40
42	Implementación de fachadas	4 días	sáb 05/05/07	vie 11/05/07	41
43	Implementacion de capa controlador	5 días	vie 11/05/07	jue 17/05/07	42
44	Implementación de las acciones	5 días	vie 11/05/07	jue 17/05/07	
45	Implementación de la vista	8 días	jue 17/05/07	lun 28/05/07	44
46	Creación de las paginas jsp	6 días	jue 17/05/07	vie 25/05/07	
47	Configuración de struts y otras configuraciones	1 día	vie 25/05/07	sáb 26/05/07	46
48	Reunión seguimiento del proyecto	1 día	sáb 26/05/07	lun 28/05/07	47
49	Pruebas	11 días	mar 29/05/07	mar 12/06/07	48
50	Pruebas individuales	5 días	mar 29/05/07	lun 04/06/07	
51	Pruebas de unidad	6 días	lun 04/06/07	mar 12/06/07	50
52	Fin de la segunda iteración	0 días	mar 12/06/07	mar 12/06/07	51
53	Reunión seguimiento del proyecto	0 días	mar 12/06/07	mar 12/06/07	52
54	Desarrollo de la memoria del proyecto	20 días	mar 12/06/07	lun 09/07/07	53
55	Reunión seguimiento del proyecto	0 días	lun 09/07/07	lun 09/07/07	54

4 Desglose de las tareas planificadas por fechas y su duración en días

Para la primera iteración se hizo una estimación de 58 días en total y para la segunda y última iteración 54 días. La diferencia entre una iteración y otra se intento plantear pensando que la primera iteración ofrecería mas trabajo tanto por ser la primera como por poner en práctica los nuevos conocimientos, sabiendo que hasta no tener experiencia la

velocidad de trabajo será inferior frente a la velocidad que se puede desempeñar con experiencia.



5 Diagrama de Gantt de la planificación realizada

Como condiciones especiales para esta planificación inicial comentar que el conocimiento de las tecnologías a utilizar en el proyecto es escaso. Se planifican unos días para el aprendizaje y así evitar lo máximo posible la mala planificación posterior.

Los gastos estimados para el proyecto son los siguientes:

Analista/Programador: 784 Horas x 15€/Hora= 11760€

Jefe de proyecto : 20 Horas x 15€/Hora= 300€

15% Gastos varios : 1819€

TOTAL : 13879€

5.2. Evaluación de la planificación realizada

Una vez finalizado el proyecto, observamos si la planificación realizada ha sido correcta o no.

Se han cumplido tanto la fecha de inicio como la fecha de fin de proyecto, pero las fechas intermedias han sufrido bastantes cambios respecto a la planificación. La principal causa de estos cambios ha sido la poca madurez en las tecnologías utilizadas. Se ha ido produciendo una mejora constante en la velocidad a lo largo del desarrollo, llegándose a rebajar en más de la mitad el tiempo la realización de tareas similares el principio y al final del proyecto.

En resumen, la primera iteración se ha prolongado hasta los 72 días, dejando solo 40 días para la segunda y última iteración. Como se ha cumplido con la fecha final del proyecto, los gastos totales a penas han sufrido cambios respecto a los planificados.

6. Requisitos del Sistema

6.1. Introducción

En nuestra aplicación se podrá diferenciar tres tipos de usuarios, usuario registrado, usuario sin registrar y usuario administrador.

Un usuario se registra mediante un formulario donde deberá especificar un pseudónimo (login) y una contraseña, y como información personal, su nombre, apellidos, fecha de nacimiento, sexo, correo electrónico, teléfono, código postal, idioma y país. Nótese que toda la información personal debe ser real por motivos de concordancia con, por ejemplo, los datos de las tarjetas de crédito o cuentas bancarias. Solo un usuario administrador puede registrar un nuevo administrador, siendo la información a cumplimentar la misma que en un registro de usuario normal. En cualquier momento, el usuario podrá cambiar datos del registro, a excepción de su nombre, apellidos y fecha de nacimiento. Después de que el nuevo usuario esté registrado, podrá identificarse con su pseudónimo y contraseña, pudiendo seleccionar una opción para recordar la contraseña para futuras sesiones y no tener que escribirla de nuevo. El usuario podrá salir explícitamente, en este caso ya no se recordará la contraseña si previamente había activado la opción.

Una vez que el usuario se ha identificado, puede acceder a un menú de su cuenta. Entre las opciones de este menú, se encuentra la del registro de tarjetas bancarias. Para introducir una nueva tarjeta deberá indicar el tipo de tarjeta, el número de tarjeta, la fecha de caducidad (no se puede introducir una fecha anterior a la actual) y el número de verificación. El titular de la tarjeta vendrá dado por el nombre introducido en el formulario de registro del usuario, de ahí una de las necesidades de dar datos reales. Si el registro de tarjeta es correcto, se añadirá a la lista de tarjetas bancarias ya registradas. El número de tarjetas registradas no tiene límite y en cualquier momento puede eliminar la que quiera.

Otra opción del menú es el ingreso de dinero en su cuenta. La cuenta del usuario es creada automáticamente cuando se registra. El ingreso mínimo será de diez euros, se elegirá la tarjeta a través de la cual realizar el ingreso y en caso de no poseer ninguna tarjeta, se solicitará al usuario que registre al menos una. Si el usuario quiere retirar dinero de su cuenta hacia una cuenta bancaria, tendrá una opción para ello y deberá cumplimentar un formulario

donde se le solicitará el nombre y dirección del banco, el número y tipo de cuenta, el código swift, instrucciones adicionales y por último la cantidad que desea retirar. Continuando con el menú de usuario, una opción será la de cambiar preferencias de usuario, en un principio solo existirá una, que será el sistema de apuestas y se podrá elegir entre el sistema decimal, americano o fracción. Nótese que en el momento en que un usuario se registra, se asignarán valores por defecto a las preferencias existentes. Por último tenemos dos opciones para consultar la historia de apuestas y la historia financiera. En ambos casos se podrán realizar las búsquedas por fecha inicial y final o bien especificando un periodo de tiempo. Para el histórico de apuestas la búsqueda se filtrará por el estado de la apuesta (pendiente, ganada, perdida o en espera) y en el histórico financiero por tipo de operación (ingreso, retiro, ingreso por apuesta o retiro por apuesta).

Cuando el usuario que se identifica es un administrador, podrá acceder a otro menú, en este caso con las opciones de administrador. La primera de ellas será el acceso al árbol de categorías, eventos y tipos de eventos. El recorrido será parecido al de un usuario normal cuando mira las apuestas existentes. Para cada categoría, el administrador podrá borrarla, esto conlleva a borrar las categorías que cuelgan de ella y sus eventos. Cualquiera de los caminos que lleven a un tipo de apuesta abierta no se borrará. Los tipos de apuesta tampoco se borrarán nunca. Por otro lado podrá crear una nueva subcategoría, para ello debe especificar un nombre y la pregunta por defecto, esta pregunta sirve para que en el momento de mostrar los tipos de apuesta que pertenezcan a eventos de esta categoría, inicialmente solo se vean los tipos de apuesta con esa pregunta. Otra función más sobre una categoría es crear un evento, que puede ser “Custom” o específico de un deporte que posea un “Wizard” para su creación. Para un evento Custom se deberá indicar un nombre, y en un principio solo existirá un ‘Wizard’ para partidos de fútbol de liga española de primera división, en cuya creación se deberá indicar además del nombre del evento, los dos equipos participantes. Para la elección de los equipos se accede a la base de datos de información deportiva para ofrecer unos desplegables con todos los equipos a escoger. Para un evento, el administrador podrá crear los tipos de apuesta que quiera, indicando el nombre del evento, su pregunta, el estado inicial y la fecha de comienzo del tipo de apuesta, a continuación añadirá las opciones, indicando el nombre de cada opción y su probabilidad, a partir de la cual se crea la cuota. Para los tipos de apuesta que estén creados, el administrador puede ver el estado en el que está, con la opción de cambiar dicho estado según el criterio de que si una apuesta está cerrada, terminada o cancelada, no podrá abrirla y no podrá ser terminada si no está cerrada

o cancelada. Las apuestas sobre ese tipo de apuesta solo podrán realizarse hasta que el administrador la cierre o bien hasta que se pase la fecha de comienzo del tipo de apuesta, en este caso, al administrador se le indicará que el tipo de apuesta se ha cerrado. Volviendo al menú del administrador, existe la opción para crear preguntas, para ello debe indicar una descripción, un comentario y el número de respuestas. Por otro lado podrá hacer una completa gestión de la información deportiva, ahora solo de fútbol, pudiendo añadir, modificar, borrar o cambiar de equipo a un jugador, y añadir, borrar o modificar un equipo de fútbol. Como se comentó antes, otra opción le permite registrar a otros usuarios. Por último, el menú le permitirá la creación de una nueva preferencia, indicando el nombre y los distintos valores que pueda tener, para cada valor se indicia un valor numérico y otro textual, una vez creado se asignará uno por defecto a todos los usuarios registrados.

Todos los usuarios podrán navegar por todas las categorías, eventos y tipos de apuesta. Cuando se accede a una categoría, si tiene eventos, se mostrarán junto con su tipo de apuesta que tenga la pregunta igual a la pregunta por defecto de la categoría. Se dará una opción en cada evento para mostrar todos sus tipos de apuesta sin importar la pregunta o bien elegir una pregunta en concreto que el usuario prefiera. Si la categoría además posee subcategorías, se mostrarán junto con sus subcategorías y eventos, pero estos sin los tipos de apuesta.

Cuando el usuario quiera añadir una opción de apuesta al talón, pinchará sobre un enlace y se añadirá. Cuando se decida a apostar o simplemente quiera observar y gestionar su talón, pinchará en ver talón, ahora podrá borrar opciones una a una o todas a la vez, y refrescar el talón. Solo un usuario identificado puede realizar apuestas simples de las opciones que quiera o una apuesta combinada de todas las opciones presentes en el talón. Si solo hay una opción en el talón, o más de una opción de un mismo tipo de apuesta, no se podrá realizar apuestas combinadas. La apuesta mínima será de un euro y las ganancias máximas de mil euros. Cuando el usuario realice la apuesta final, esta se hace persistente y se mostrará un ticket con información de la operación, a su vez se creará una operación de pérdida en apuesta. Nótese que el talón y las opciones incluidas en él no son persistentes.

En cuanto a los estados de los tipos de apuesta, existen cuatro estados. Si el tipo de apuesta está abierto, el usuario podrá realizar apuestas. Si se pausa la apuesta por algún motivo, al usuario se le indicará textualmente y no se le permitirá apostar. Si se cierra, el tipo de apuesta desaparecerá a la vista del usuario. Por último, si el tipo de apuesta se cancela por

algún motivo, deberá reembolsarse a todos los usuarios que hayan echo apuestas simples el dinero que hayan apostado, se creará un movimiento registrando dicha acción y se comprobarán las apuestas combinadas, en este caso, si era la última apuesta que quedaba para ganar la combinada, se dará por ganada y el estado de la apuesta (la cancelada) será puesto a cancelada, mientras que si aun quedan mas apuestas para ganar la combinada, solo se pondrá como cancelada, que a vistas de comprobaciones posteriores se contará como ganada. Una vez que el tipo de apuesta ya tiene resultados, el administrador será quien se encargue de seleccionar la opción ganadora y el tipo de apuesta pasará al estado de finalizado. Con la opción ganadora ya marcada, se buscan los ganadores y perdedores del tipo de apuesta, tanto para las apuestas simples creadas sobre él como para las posibles combinadas en la que se incluya una opción de este tipo de apuesta. Se registrará una operación de ingreso por apuesta para los ganadores y se le ingresarán los beneficios pertinentes.

6.2. Actores

Se puede identificar tres tipos de usuarios con diferentes privilegios y opciones dentro de nuestra aplicación. A continuación se exponen los diferentes actores comentado brevemente las diferencias.

Usuario no autenticado: Se trata de la persona que accede a la aplicación, pero no se ha autenticado, ya sea por el formulario de autenticación o bien realizando nuevo registro de usuario. Este tipo de usuarios podrán tener un holgado acceso a diferentes partes de la aplicación, de forma que pueda apreciar como funciona realmente sin tener que registrase e identificarse. Entre otros podrá navegar libremente por las categorías y eventos, podrá manejar su propio talón, con la diferencia de que no se le permite realizar apuestas.

Usuario registrado y autenticado: Cuando la persona que ha entrado en la aplicación se autentica, pudiendo realizar las funciones de usuario para las que fue pensada la aplicación, como apostar y acceder a todas sus opciones como usuario registrado y autenticado.

Usuario administrador: El usuario administrador será aquel que se autentique y su usuario sea de tipo administrador. Podrá acceder a su menú de administrador y gestionar las

categorías, eventos, tipos de apuesta, información deportiva, etc. A su vez podrá realizar todas las tareas posibles para un usuario registrado y autenticado no administrador.

6.3. Casos de Uso

En la aplicación se pueden diferenciar los casos de uso según los actores que pueden hacer uso de ellos. En un primer lugar habrá unas funcionalidades exclusivas del usuario administrador, luego habrá otros casos de uso destinados al usuario registrado y autenticado. Por último las funcionalidades que comparten los tres tipos de actores. Comentar que todas las funcionalidades de un usuario registrado y autenticado también lo son para el usuario administrador.

A continuación se comentan los casos de uso que implementa la aplicación, especificando su objetivo y que actores hacen uso de dicha funcionalidad.

Registro de usuario: La aplicación permite el registro de nuevos usuarios. Como caso especial, comentar que el registro de un usuario administrador solo es posible por parte de un usuario administrador ya registrado y autenticado.

Navegar por las categorías y eventos: Funcionalidad que permite visualizar las categorías junto con sus eventos y subcategorías. Para los eventos se mostrarán los tipos de apuesta existentes. Cualquier usuario puede realizar el recorrido por categorías.

Gestión del talón de apuestas: Cualquier usuario podrá introducir nuevas opciones al talón, borrar opciones, ya sean una a una o todas a la vez, y refrescar el talón en busca de cambios en el estado del tipo de apuesta de la opción.

Realizar apuestas simples y combinadas: Mediante el talón, donde estarán las opciones escogidas por el usuario, se introduce la cantidad deseada para apostar y la apuesta se hace persistente. Sólo es posible para usuarios registrados y autenticados.

Visualizar historial de apuestas: Mediante el filtrado por el estado de apuesta y por fecha o periodo, se mostrarán las apuestas que cumplan las opciones de filtrado junto con sus detalles. Destinado únicamente al usuario registrado y autenticado.

Visualizar historial de apuestas: Mediante el filtrado por tipo de movimiento y fecha o periodo, se mostrarán los movimientos financieros que satisfagan las condiciones de filtrado. Cada uno irá acompañado de sus detalles. Destinado al usuario autenticado.

Realizar ingreso desde tarjeta: Se podrá escoger una de las tarjetas previamente registradas y poner el importe que se quiere transferir desde la tarjeta a su cuenta en la aplicación. Sólo permitido a usuarios registrados y autenticados.

Gestionar tarjetas de crédito: Posibilidad de añadir nuevas tarjetas o eliminar las previamente registradas. No existe límite de tarjetas por usuario. Los usuarios registrados y autenticados serán los únicos que puedan hacer uso de esta funcionalidad.

Retirar dinero a cuenta bancaria: Se podrá retirar dinero de la cuenta de la aplicación a una cuenta bancaria mediante un sencillo formulario. Sólo para usuarios registrados y autenticados.

Cambiar preferencias: Mediante un menú donde aparecerán las diferentes preferencias junto con los valores posibles para cada una. Destinado a usuarios registrados y autenticados.

Marcar opción de un tipo de apuesta como ganadora: Cuando un tipo de apuesta está cerrado y ya hay opción ganadora, esta se marcará como ganadora y se realizarán los pagos automáticamente. Este caso de uso es exclusivo del usuario administrador.

Gestionar estados de las apuestas: Siguiendo unos criterios lógicos, se podrá realizar el cambio de estado de los tipos de apuesta. Esta funcionalidad es única del administrador.

Cancelar apuesta: Se trata de un caso especial de cambio de estado de apuesta. En este caso, se realizan automáticamente las gestiones derivadas de la cancelación de un tipo de apuesta. Exclusivo de usuario administrador.

Gestionar de categorías: Se podrán crear nuevas categorías o subcategorías de otras ya existentes. Por otro lado se pueden eliminar categorías con restricciones automáticas tales como no poder eliminar categorías que incluyan tipos de apuestas todavía sin finalizar. Exclusivo de administrador.

Gestionar de preguntas: Se podrá crear nuevas preguntas. Sólo para usuario administrador.

Gestionar de eventos: Funcionalidad que permite crear nuevos eventos en las categorías existentes, así como eliminarlos. Exclusivo de usuario administrador.

Creación de tipos de apuestas: Este caso de uso tiene como objetivo la creación de tipos de apuesta para los eventos existentes, pudiendo marcar una fecha para su cierre automático. A la vez que se crea un tipo de apuesta, se crean sus opciones. Función exclusiva de usuario administrador.

Uso de wizards para deportes con información deportiva: En la creación de eventos se puede hacer uso los ayudantes disponibles para la creación de eventos específicos. Uso específico del administrador.

Gestionar información deportiva: Se podrán crear nuevos equipos de fútbol, actualizar la información de aquellos ya creados o eliminarlos. En cuanto los jugadores de fútbol, se podrán crear nuevos jugadores en los equipos de fútbol, y posteriormente actualizar su información o eliminarlo. Al actualizar su información se le puede cambiar de equipo.

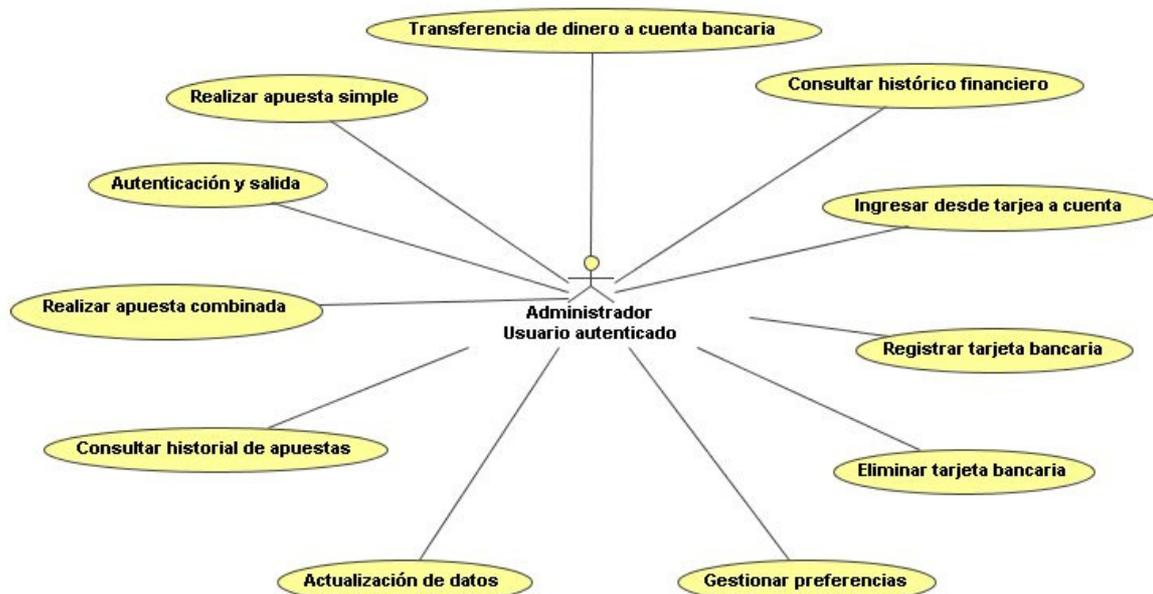
6.4. Modelo de Casos de Uso

A continuación se muestran los diagramas con los diferentes actores y sus casos de uso.



6. Diagrama de casos de uso del administrador

Usuario administrador y usuario registrado y autenticado junto a los casos de uso que comparten.



7. Diagrama de casos de uso del usuario registrado y autenticado

Los tres actores diferentes junto con los casos de uso de carácter general que comparten.



8. Diagrama de los casos de uso generales para cualquier usuario

7. Diseño de la aplicación

Para la realización de la aplicación se han seguido una serie de patrones que proporcionan al producto robustez, fiabilidad y modularidad. A continuación se comentan los principales patrones utilizados [15].

- **Patrones arquitectónicos**

Tienen como objetivo aconsejar la arquitectura global que debe seguir una aplicación.

Patrón Model View Controller (MVC): Patrón que proporciona una clara separación entre la lógica de negocio (modelo) y la interfaz gráfica con el usuario (vista). Por otro lado está un controlador que proporciona la interfaz entre la vista y el modelo. Este patrón favorece en gran medida la reutilización de clases, por ejemplo, para una misma lógica de negocio se pueden utilizar diferentes vistas. Por otro lado, junto con otros patrones de diseño, se consigue que los cambios en diferentes capas no sean apreciados por el resto.

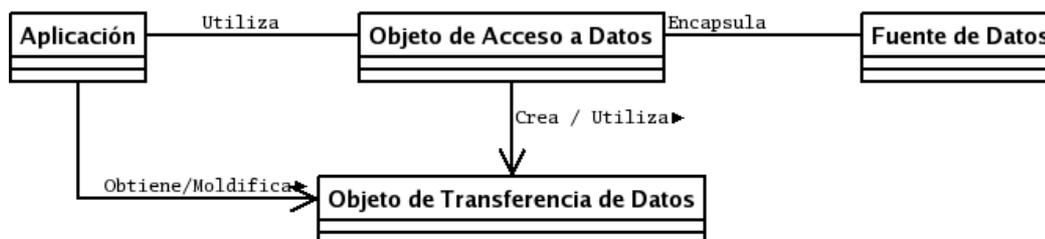
Patrón Layers: Ofrece soporte para la arquitectura modelo vista controlador. La aplicación se estructura en diferentes capas, ofreciendo transparencia hacia la implementación de cada capa. Esto favorece la separación de roles en la creación del producto software. En nuestro caso, la persona encargada del diseño gráfico de la aplicación a penas debe tener conocimientos de java. En esta aplicación, dentro de las tres capas definidas por el modelo MVC, se aplica a su vez el patrón Layers definiendo diferentes paquetes.

- **Patrón de diseño**

Explican como resolver un problema concreto de diseño.

Patrón Value Object (VO): Actualmente se ha cambiado el nombre a Transfer Object (TO). Representan el estado de los objetos de dominio (persistentes). Para ello tiene atributos privados y como solo métodos get y set para los atributos modificables. También puede tener un método toString e implementarán la interfaz Serializable en caso de tener que enviarse por la red.

Patrón Data Access Object (DAO): Patrón que permite desacoplar la lógica de negocio del acceso a la base de datos, ocultando el tipo de repositorio de datos utilizado. El interfaz DAO abstrae las operaciones sobre la fuente de datos, definiendo operaciones para insertar, borrar, actualizar y recuperar los objetos persistentes de un tipo. Para cada fuente de datos concreta se ofrecerá una implementación de la interfaz.



9. Diagrama del patrón DAO

Patrón Page-By-Page Iterator: Perfecciona el acceso a una lista grande Value Objects aumentando la eficiencia. A la Vista le ofrece una forma de mostrar solo una cantidad de objetos y no todos los de la lista, mostrando los objetos por rangos.

Patrón Session Facade: Patrón cuyo objetivo principal es el control de flujo. Se encarga de ofrecer una API sencilla con un conjunto de casos de uso lógicamente relacionados proporcionando un método por cada caso de uso.

Patrón Budines Delegate: El objetivo de este patrón es ocultar las tecnologías utilizadas en el modelo, separando la lógica de negocio del código que la usa. De esta forma, este patrón maneja la complejidad de los componentes del modelo y se encarga del manejo de excepciones para conseguir un interface de los componentes de negocio simple para el controlador. En nuestro caso, Session Facade y Business Delegate se van a integrar en uno sólo.

Patrón Factory: Patrón que permite instanciar objetos de la misma familia, esto es, que implementen la misma interfaz, sin que le código dependa de los nombres concretos de las clases. En nuestro caso, se utilizará este patrón para implementaciones concretas de nuestros DAOs y de las fachadas. Mediante un fichero de configuración, se lee el nombre de la clase concreta que se quiere instanciar y mediante la API de Java (`java.lang.Class`) se carga la clase solicitada. En el caso de los DAOs, permite cambiar de repositorio de datos

simplemente cambiando la el nombre en la configuración para que instancie la nueva implementación.

Patrón FrontController: Patrón propio de la capa controlador de nuestra aplicación. Proporciona un punto de acceso centralizado para manejar las peticiones (request) de la capa vista. Para ello se utiliza un solo servlet (Front-Controller) al que se redirigirán todas las peticiones de la aplicación. Mediante un fichero de configuración, el controlador sabrá como responder ante las peticiones, de forma que para una petición redirige el flujo de control hacia la clase correspondiente que se debe ejecutar. Las clases Action del controlador recogen los valores de los parámetros de la petición, posiblemente mediante JavaBeans que extienden ActionForm, se invocará al modelo que devolverá los resultados que se dejarán de nuevo en la request y por último se dirigirá a la vista al lugar correspondiente según el resultado obtenido en toda la operación. Struts proporciona una implementación de este patrón que facilitará el desarrollo de la capa controlador de la aplicación.

7.1. Arquitectura general

La arquitectura JEE implica un modelo de aplicaciones distribuidas en diversas capas o niveles. Esta idea proporciona independencia entre las diferentes capas ofreciendo fiabilidad, mantenibilidad, flexibilidad.

Siguiendo el patrón MVC, la arquitectura global tendrá tres capas bien diferenciadas:

Modelo: En esta capa se encuentra la lógica de negocio de la aplicación. Se encarga de las transferencias entre la base da datos y la aplicación, asegurando la persistencia de los datos. Hacia el controlador ofrece una serie de fachadas que permite al controlador ejecutar las acciones que posee. Esta capa a su vez sigue el patrón arquitectónico Layers.

Controlador: Se puede decir que es la capa que une el modelo y la vista, recibiendo las peticiones de la vista y llamando al modelo para realizar la acción pertinente. A continuación recibe la información que el modelo le proporciona y la envía de forma adecuada a la vista.

Vista: Se puede considerar la interfaz con el usuario, permitiéndole realizar peticiones y mostrándole la información devuelta a través del controlador y modelo.

7.2. Subsistema 1

Se trata del subsistema principal de la aplicación.

7.2.1. Objetivos

El objetivo de este subsistema es la implementación de las tres capas del modelo MVC utilizado como arquitectura global.

7.2.2. Arquitectura

Este subsistema está organizado en tres paquetes principales a partir del paquete global `es.apuestasinparar`, de forma adecuada para conseguir diferenciar la capa modelo, vista y controlador.

es.apuestasinparar.modelo: Comprende la toda la capa modelo de la aplicación. Contiene los subpaquetes donde se implementan las fachadas del modelo y los subpaquetes donde se implementan las clases persistentes. Los detalles de estos paquetes se realizarán en capítulos posteriores.

es.apuestasinparar.http: Paquete que abarca la capa vista y controlador del subsistema. Contiene a su vez los siguientes subpaquetes.

- `es.apuestasinparar.http.controller:` Paquete donde se define la capa controlador.
- `es.apuestasinparar.http.view:` Paquete donde se define la capa vista de la aplicación.



10. Arquitectura en paquetes de la aplicación

7.2.3. Modelo

7.2.3.1. Clases Persistentes

- **Modelo de objetos de dominio**

En la imagen se muestran los objetos persistentes de la aplicación en un diagrama dónde se aprecian las relaciones lógicas entre ellos.

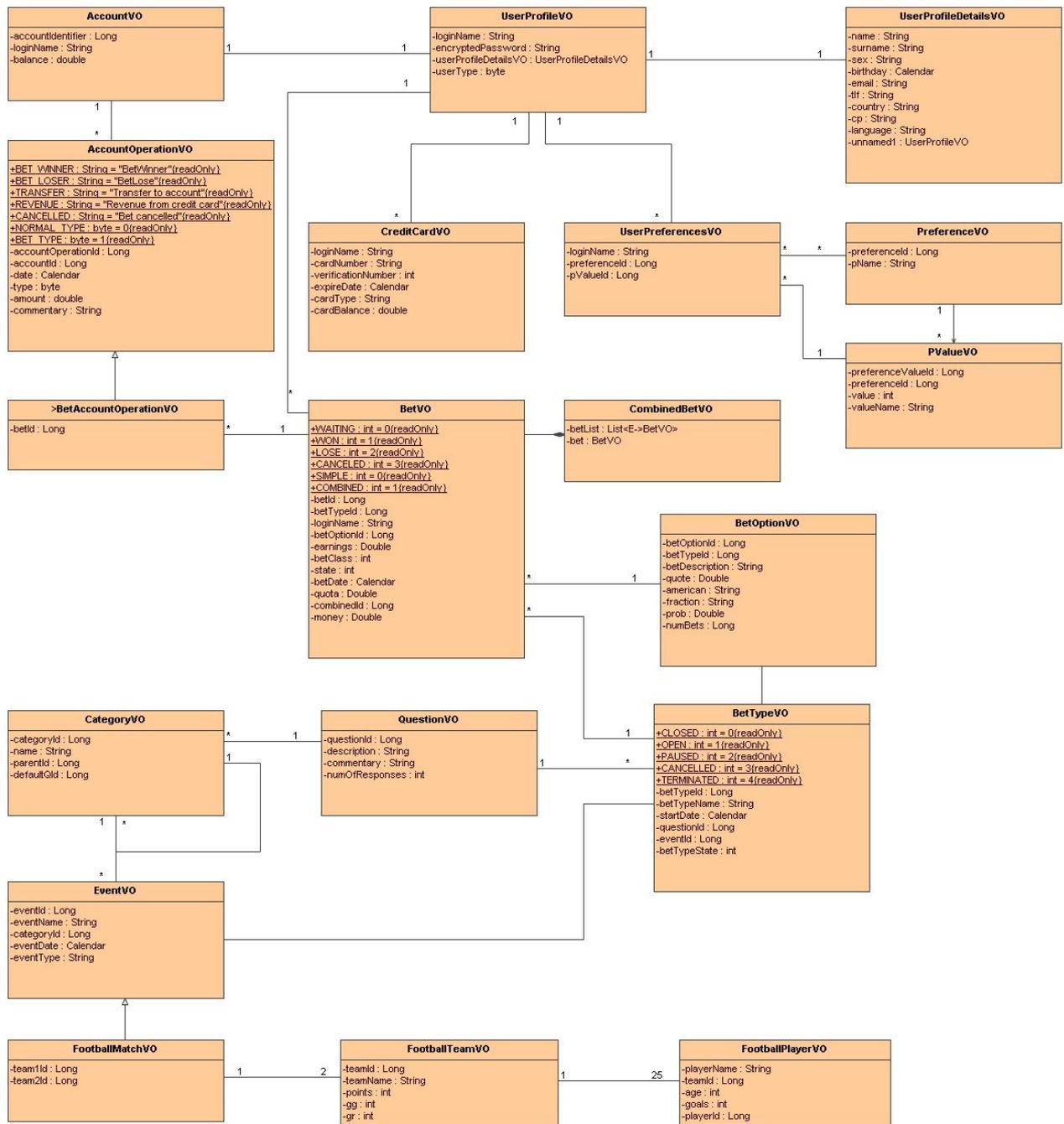
UserProfileVO, que representa a un usuario, puede estar relacionado con un número no limitado de CreditCardVO y UserPreferenceVO, tarjetas bancarias y preferencias respectivamente. Esta relación será a través del atributo loginName. Un usuario solo puede tener una cuenta, por lo que la relación entre UserProfileVO y AccountVO será de uno a uno. A su vez AccountVO, podrá estar relacionado con infinitos movimientos bancarios, representados por el objeto AccountOperationVO el cual posee como atributo el identificador de la cuenta sobre la cual se realizó el movimiento.

El objeto UserPreferenceVO está relacionado con una preferencia y un valor de esa preferencia, a través de sus respectivos identificadores. Los valores de preferencias, representados por PValueVO tienen un atributo con el identificador de la preferencia a la que pertenece.

Las apuestas, representadas por el objeto BetVO están relacionadas con un tipo de apuesta, BetTypeVO, y una opción sobre ese tipo de apuesta, representado por BetOptionVO. Estas referencias se mantienen a través de los identificadores de los tipos de apuesta y la opción. Un tipo de apuesta puede tener un número no finito de options.

Para crear el árbol de categorías, eventos y tipos de apuesta, el objeto EventVO tiene el identificador de la categoría a la que pertenece y el tipo de apuesta a su vez tiene el identificador del evento al que pertenece. Una categoría puede tener subcategorías, para ello el objeto CategoryVO tiene como atributo el identificador de su categoría padre.

La última relación a destacar es la existente entre el objeto FootballMatchVO, que representa un evento concreto de partido de fútbol y tiene los identificadores de los dos equipos enfrentados y estos a su vez los identificadores de sus respectivos jugadores.



11. Diagrama de clases de los objetos persistentes de la aplicación

- VOs

Los objetos del dominio están implementados siguiendo el patrón Value Object. A continuación se detallará cada una de las clases persistentes de la capa modelo.

UserProfileVO: Clase que representa a un usuario registrado. Contiene un UserProfileDetailsVO donde se encapsula la información detallada del usuario, mientras que en UserProfileVO están los siguientes atributos:

- loginName: Pseudónimo del usuario con el que se autenticará.
- encryptedPassword: Contraseña encriptada del usuario.
- userProfileDetailsVO: Información detallada y personal del usuario.
- userType: Tipo de usuario, pudiendo ser usuario normal o usuario administrador.

AccountVO: Representación de la cuenta que tiene cada usuario registrado. Una cuenta tiene como atributos:

- accountId: Identificador único para la cuenta.
- loginName: String para el pseudónimo del usuario al que pertenece la cuenta.
- balance: Dinero del que dispone la cuenta.

AccountOperationVO: Representación de un movimiento financiero en la cuenta del usuario. Dispone de los siguientes atributos:

- BET_WINNER, BET_LOSER, TRANSFER, REVENUE, CANCELLED: Son cadenas constantes para expresar el comentario que contiene una operación.
- INGRESO, RETIRO, BET_WON, BET_BETTING: Constantes para expresar el tipo de movimiento del que se trata.
- accountOperationId: Identificador único de cada operación.
- accountId: Identificador de la cuenta sobre la cual se ha realizado el movimiento.
- type: Tipo de operación.
- amount: De tipo double, es la cantidad dinero que se maneja en la operación.
- commentary: Comentario adicional sobre la operación.

BetAccountOperationVO: Clase que hereda de AccountOperationVO y que representa un movimiento en la cuenta generado por una apuesta. Además de los atributos heredados, posee los siguientes:

- betId: Identificador de la apuesta asociada a este movimiento.

CreditCardVO: Representación de una tarjeta de crédito de un usuario. Sus atributos son los siguientes:

- loginName: String del pseudónimo del usuario que ha registrado dicha tarjeta.
- cardNumber: Long para el número de tarjeta de crédito que es único para cada tarjeta.
- expireDate: De la clase Calendar, representa a fecha de caducidad de la tarjeta.
- cardType: Int para expresar los diferentes tipos de tarjetas admitidos.
- cardBalance: Dinero del que dispone la tarjeta.

UserPreferencesVO: Preferencia de un usuario, con referencia a la preferencia con el valor elegido por el usuario para esa preferencia. Cuenta con los siguientes atributos:

- loginName: Pseudónimo del usuario al que pertenece.
- preferenceId: Identificar de la preferencia a la que hace referencia.
- pValueId: Identificar del valor de la preferencia.

PreferenceVO: Preferencia que crea el administrador. Sus atributos son los siguientes:

- preferenceId: Identificar único de la preferencia.
- pName: Nombre de la preferencia.

PValueVO: Se trata del valor de una preferencia. Los atributos que posee son:

- preferenceValueId: Identificador único del valor.
- preferenceId: Identificador de la preferencia a la que pertenece este valor de preferencia.
- value: Representa el valor en si.
- valueName: Nombre para el valor.

BetVO: Representa la apuesta persistente que se crea cuando un usuario finaliza el proceso de apuesta. Sus atributos son los siguientes:

- WAITING, WON, LOSE, CANCELED: Constantes para la representación del estado de la apuesta.

- SIMPLE Y COMBINED: Constantes para representar el tipo de apuesta que es.
- BetId: Identificador único de la apuesta.
- BetTypeId: Identificador del tipo de apuesta al que pertenece.
- LoginName: String con el pseudónimo del usuario al que pertenece la apuesta.
- BetOptionId: Identificar de la opción de apuesta que el usuario ha elegido.
- Earnings: Double que representa las ganancias posibles con la apuesta.
- BetClass: Clase de apuesta, simple o combinada.
- State: Estado de la apuesta, ganada, perdida, en espera o cancelada.
- BetDate: Del tipo Calendar, representa la fecha y hora en la que ha sido creada la apuesta.
- Quota: La cuota de la opción de apuesta en el instante de la creación de la apuesta.
- CombinedId: En caso de ser una apuesta perteneciente a un grupo de apuestas de una combinada, este identificador es el de la apuesta que representa la apuesta combinada.

BetOptionVO: Se trata de una opción para un único tipo de apuesta. Sus atributos son los siguientes:

- BetOptionId: Identificador de la opción.
- BetTypeId: Identificador del tipo de apuesta al que pertenece.
- BetOptionDescription: String para la descripción de la opción.
- Prob: Double que representa la probabilidad de la opción, esta probabilidad condiciona la cuota.
- Quote, Fraction y American: Las tres posibles representaciones de la cuota asignada a esta opción.

BetTypeVO: Representa un tipo de apuesta que pertenecerá a un evento. Sus atributos son:

- CLOSED, OPEN, PAUSED, CANCELED y TERMINATED: Constantes que representan los diferentes estados en los que puede estar el tipo de apuesta, serán utilizados en las

distintas clases que creen un nuevo tipo de apuesta para asignar un valor al atributo `betTypeState`.

- `BetTypeId`: Identificador único del tipo de apuesta.
- `BetTypeName`: Nombre del tipo de apuesta.
- `StarDate`: Del tipo `calendar`, señala la fecha límite para poder realizar apuestas sobre este tipo de apuesta.
- `QuestionId`: Identificador de la pregunta con la que está relacionado.
- `EventId`: Identificador del evento al que pertenece.

QuestionVO: Denominado como pregunta, se puede decir que representa el modelo de tipo de apuesta, concretando algunos parámetros comunes a todos los tipos de apuesta que compartan esta pregunta. Sus atributos son los siguientes:

- `QuestionId`: Identificador único de la pregunta.
- `Description`: String para una pequeña descripción de la pregunta, se podrá ver a la hora de mostrar un tipo de apuesta con esta pregunta.
- `Commentary`: Al igual que la descripción, es un String que se verá esta vez al pie del tipo de apuesta.
- `NumOfResponses`: Número de respuestas común para dicha pregunta, esto es, el número de opciones que debería tener.

CategoryVO: En conjunto, las categorías forman un árbol, donde una categoría puede ser padre de una o varias categorías. Cada `CategoryVO` representa una categoría en concreto que estará relacionada con una categoría padre. Sus atributos son los siguientes:

- `CategoryId`: Identificador único para la categoría.
- `Name`: Nombre de la categoría.
- `ParentId`: Identificador de la categoría padre.
- `DefaultQId`: Identificador de la pregunta por defecto, será utilizado a la hora de mostrar los tipos de apuesta de los eventos de esta categoría.

EventVO: Representa un evento relacionado a una categoría en concreto. Es un evento genérico a diferencia de `FootballMatchVO`, que es concreto de fútbol. La aplicación

está preparada para la creación de eventos concretos heredando de EventVO. Los atributos son los siguientes:

- EventId: Identificador del evento.
- EventName: Nombre del evento.
- CategoryId: Long que relaciona al evento con una categoría.
- EventDate: Fecha del evento.
- EventType: Atributo que permite diferenciar los tipos de evento diferentes. De momento solo habrá que diferenciar eventos genéricos de eventos de fútbol. Este valor permite al DAO recuperar y guardar de forma adecuada la información del evento.

FootballMatchVO: Hereda de EventVO, se trata de un evento concreto de fútbol. Además de los atributos de EventVO añade los siguientes:

- Team1Id: Identificador del primer equipo involucrado en el evento.
- Team2Id: Identificador del segundo equipo del evento.

FootballTeamVO: Representa un equipo de fútbol. Sus atributos son los siguientes:

- TeamId: Identificador del equipo.
- TeamName: Nombre del equipo
- Points: Puntos que lleva en la liga
- Gg: Puntos a favor representado por un int.
- Gr: Puntos en contra

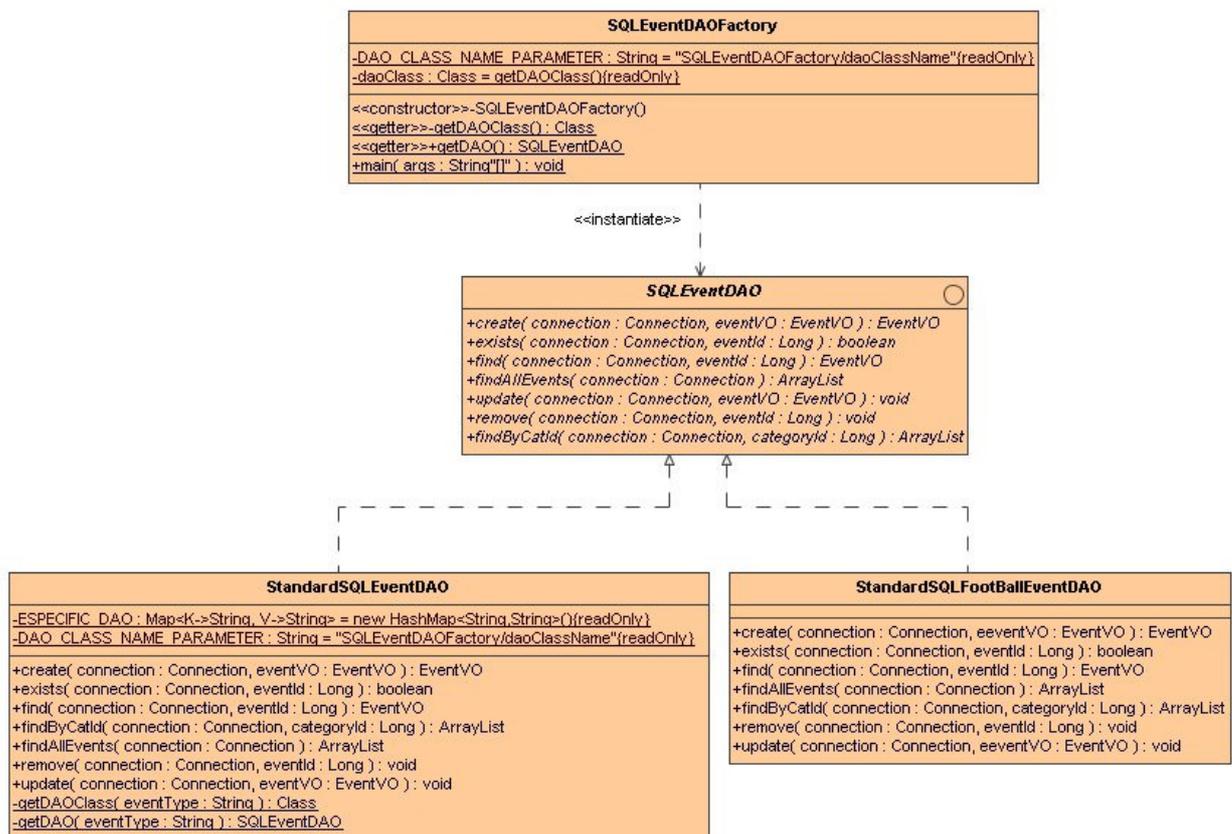
FootballPlayerVO: Representa a un jugador de fútbol y pertenece a alguno de los equipos representados por objeto anteriormente descrito. Sus atributos son los siguientes:

- TeamId: Identificador del equipo al que pertenece.
- PlayerName: String para el nombre del jugador.
- Age: Edad del jugador.
- Goals: Int para los goles del jugador.
- PlayerId: Identificador del jugador.

• DAOs

Se utiliza el patrón Data Access Object para ocultar el acceso a la base de datos, se ha implementado una solución para una base de datos relacional mediante SQL estándar.

Utilizando el patrón factoría, para cada DAO se implementa una factoría que proporcionará instancias de implementaciones concretas de la interfaz del DAO. De esta forma, se pueden realizar diferentes implementaciones para los distintos repositorios de datos que queramos usar, sin tener que realizar ningún cambio, simplemente se deberá realizar dicha implementación y cambiar el valor de la configuración de la factoría para que devuelva la instancia de la implementación deseada.



7.2.3.2. Interfaces de las Fachadas

La aplicación consta de cuatro fachadas destinadas a implementar los casos de uso. Cada fachada agrupa un subconjunto de funcionalidades, el criterio para la asignación de

dichas funcionalidades ha sido básicamente el usuario al que están orientadas así como la participación en la lógica de la aplicación.

- **Arquitectura**

Las fachadas se encuentran en cuatro paquetes diferentes

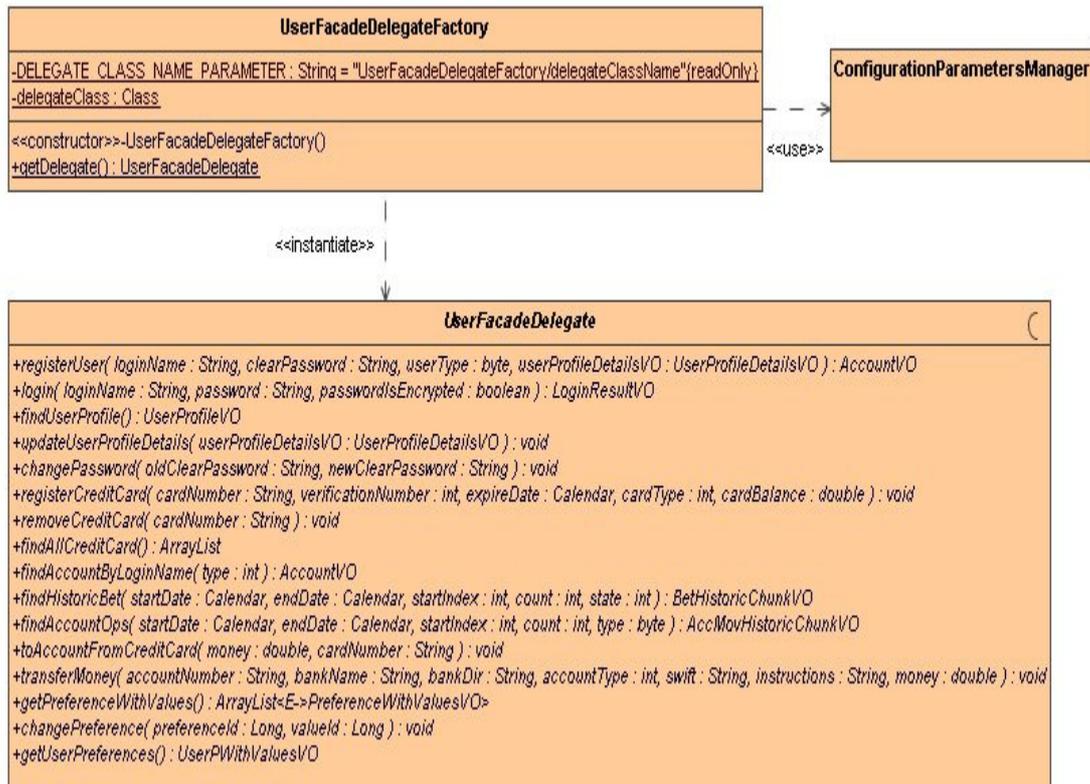
- `es.apuestasinparar.model.userfacade`
- `es.apuestasinparar.model.betfacade`
- `es.apuestasinparar.model.betcataloguefacade`
- `es.apuestasinparar.model.userfacade`

Los cuatro paquetes a su vez tienen una arquitectura similar en las cuatro fachadas. A continuación se detallará un poco más los subpaquetes junto con sus clases, para ello se usará una notación común para denotar de forma genérica una fachada: “xxxFacade”.

Paquete `es.apuestasinparar.model.xxxfacade.delegate`

En este paquete se implementa el patrón Business Delegate, permitiendo ocultar la tecnología e implementación del modelo hacia la capa vista y la capa controlador. De esta forma, los cambios de tecnologías u otros en esta capa no afectarán a la vista ni al controlador. Se implementan dos clases.

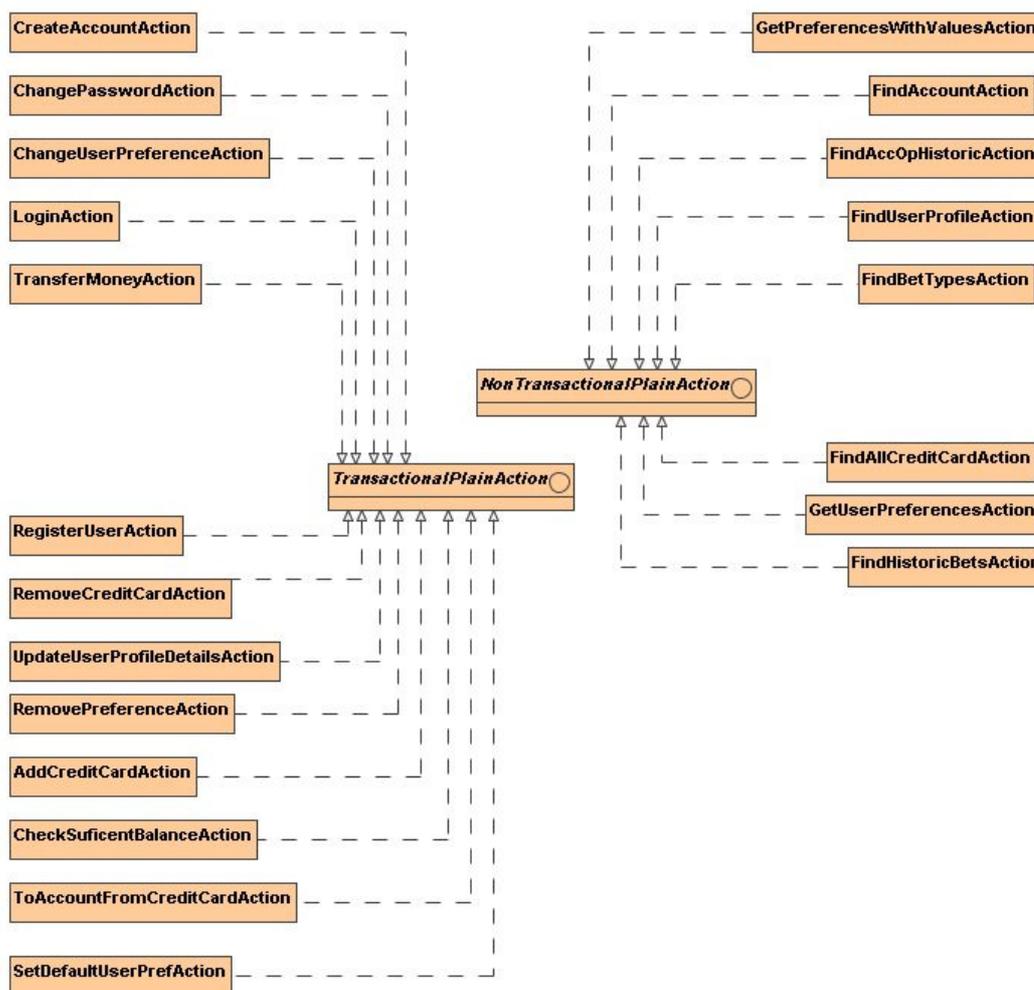
- `XxxFacadeDelegate`: Interfaz que define los métodos de la fachada. Siguiendo el patrón Session Facade habrá un método por caso de uso de la fachada.
- `XxxFacadeDelegateFactory`: Clase diseñada siguiendo el patrón Factoría. Proporciona una instancia de una implementación concreta de la interfaz anteriormente definida. Para saber que implementación concreta debe instanciar, leerá un parámetro de configuración.



Paquete es.apuestasinparar.model.xxxfacade.plain

Paquete que contiene la siguiente clase y a su vez tiene un subpaquete.

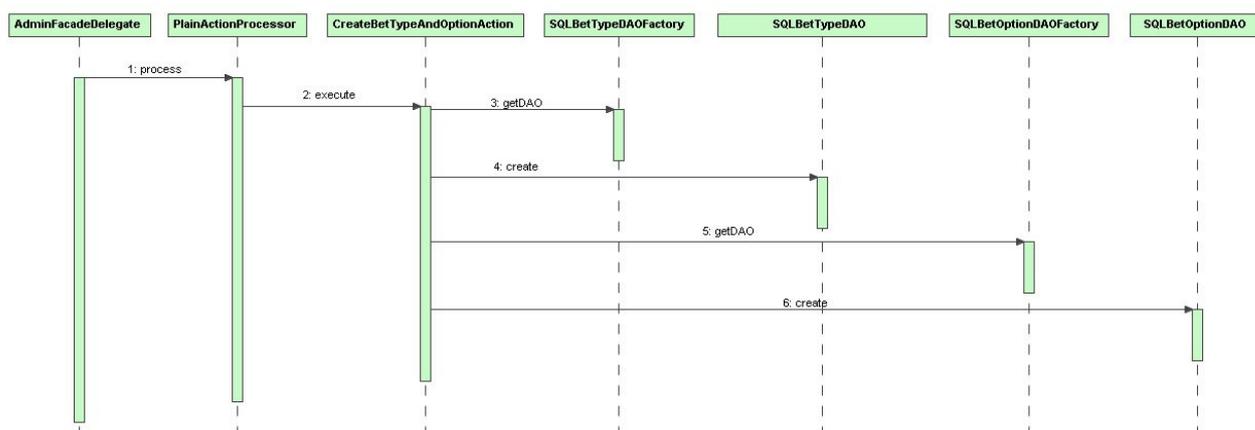
- PlainXxxFacadeDelegate: Esta clase ofrece una implementación de la interfaz de la fachada. Para evitar crear una clase excesivamente grande y con poca claridad, la implementación de cada operación del Session Facade delegará a su vez su implementación en unas clases action. Hace uso la clase DataSourceLocator para obtener el DataSource que proporciona la conexión a la base de datos con la que trabajará el interfaz DAO usado en las clases action, también hace uso de la clase PlainActionProcesor que ejecutara de forma apropiada el código de las clases action según sean transaccionales o no transaccionales.
- es.apuestasinparar.model.xxxfacade.plain.actions: En este paquete se encuentran las clases action que implementan las clases TransactionalPlainAction o bien NonTransactionalPlainAction del paquete es.apuestasinparar.util, según el tipo de operación que realice en la llamada al DAO o DAOS pertinentes para realizar la acción.



es.apuestasinparar.model.xxxfacade.vo: Contiene los Custom Object necesarios para los métodos de la fachada, son implementados de forma específica para determinados casos de uso, los cuales utilizan estos objetos para encapsular la información que debe devolver. En ocasiones existe una relación de agregación entre ellos. Estas clases siguen el patrón Value Object.

• Fachada de usuario

Se trata de una fachada con estado dado por el loginName. Su objetivo es modelar la iteración del usuario con la aplicación, es decir, contiene las funcionalidades dirigidas al usuario autenticado, a excepción del caso de uso de registro de usuario. Existen una serie de restricciones lógicas entre los métodos, de forma que si un usuario no se ha autenticado no se podrá llamar a los demás métodos, nótese que estos hacen uso del valor de loginName de la fachada y si un usuario no se autentica, la fachada no posee estado.



A continuación se detallan los métodos de la fachada, especificando su objetivo, sus parámetros de entrada y el valor o valores devueltos, detallando si es necesario el uso de un Custom Object.

registerUser: Su objetivo es realizar el registro de un nuevo usuario, así como la creación de una cuenta y la asignación de valores por defecto para las preferencias. Sus parámetros de entrada son el pseudónimo, la contraseña, el tipo de usuario y un objeto UserProfileDetailsVO que contiene la información personal del nuevo usuario. Devuelve un objeto AccountVO con la nueva cuenta creada para el usuario.

changePassword: Su objetivo es cambiar la contraseña del usuario por una nueva. Recibe como parámetros la antigua contraseña y la nueva contraseña.

changePreference: Su objetivo es cambiar el valor de una preferencia de usuario. Recibe como parámetros el identificador de la preferencia junto con el identificador del valor al cual el usuario quiere cambiar.

findAccountByLogin: Su objetivo es encontrar la cuenta de un usuario y devolver su información. Recibe como parámetro el tipo de cuenta que se quiere buscar. Devuelve un objeto AccountVO, que es la cuenta buscada. Este parámetro está pensado para la posibilidad de tener varias cuentas por usuario.

findAccountOps: El objetivo de este método es realizar la búsqueda de movimientos financieros comprendidos entre dos fechas dadas, a partir del índice de búsqueda dado y con

la restricción de la cantidad de movimientos requerido además del tipo concreto de movimiento solicitado. Recibe como parámetros una fecha de inicio, una fecha fin, un índice para el comienzo de la búsqueda, la cantidad de operaciones que se quieren buscar y el tipo de movimiento. Devuelve un objeto `AccMovHistoricChunkVO`. Este objeto es un custom object propio de la fachada y nos sirve para devolver los objeto `AccountOperationVO` requeridos junto con un boolean que indicia si existe más movimientos que cumplan los criterios de la búsqueda.

findAllCreditCard: El objetivo de este método es buscar todas las tarjetas registradas por un usuario. No recibe ningún parámetro de entrada, la búsqueda la realizar a partir de `loginName` que da estado a la fachada. Devuelve una colección de objetos `CreditCardVO`.

findHistoricBet: Su objetivo es buscar el historial de apuestas comprendidos entre dos fechas dadas, a partir de un índice de búsqueda y con la restricción de una cantidad máxima de apuestas buscadas y el estado de las apuestas. Recibe como parámetros de entrada una fecha de inicio, una fecha fin, un índice para el comienzo de la búsqueda, el estado en el que deben estar las apuestas y por último la cantidad máxima de apuestas que se quiere buscar bajo estas condiciones.

findUserProfile: Su objetivo es realizar la búsqueda de la información de un usuario. Como entrada no recibe ningún parámetro y utiliza el `loginName` de la fachada para llevar a cabo la acción. Devuelve un objeto `UserProfileVO` que contiene toda la información del usuario.

getPreferenceWithValues: Su objetivo es recuperar todas las preferencias junto con sus posibles valores. No tiene parámetros de entrada. Devuelve una colección de objetos `PreferenceWithValuesVO`. Este tipo de objetos son custom objects y cada uno representa una preferencia junto con una lista de valores posibles para ella.

getUserPreferences: Su objetivo es buscar las preferencias de un usuario junto con el valor asignado a cada una. No posee parámetros de entrada y se vale del `loginName` de la fachada para realizar la búsqueda. Como salida devuelve un custom object `UserPWithValuesVO`, que contiene una relación entre cada preferencia y su valor, representados por sus respectivos identificadores.

login: Su objetivo es autenticar a un usuario mediante su pseudónimo y su contraseña, para ello se comprueba que la contraseña sea la correcta. Como parámetros de entrada recibe el pseudónimo del usuario junto con su contraseña, además de un valor boolean que indica si la contraseña está o no encriptada. Devuelve un `LoginResultVO`, que se trata de un custom object con la información necesaria para la sesión.

registerCreditCard: Su objetivo es añadir una nueva tarjeta a la lista de tarjetas bancarias de un usuario. Sus parámetros de entrada son el número de la tarjeta, el número de verificación, la fecha de caducidad, el tipo de tarjeta y el dinero que tendrá.

removeCreditCard: Su objetivo es eliminar una tarjeta bancaria de la lista de tarjetas bancarias de un usuario. Como parámetro de entrada recibe el número de la tarjeta bancaria, el cual actúa como identificador único.

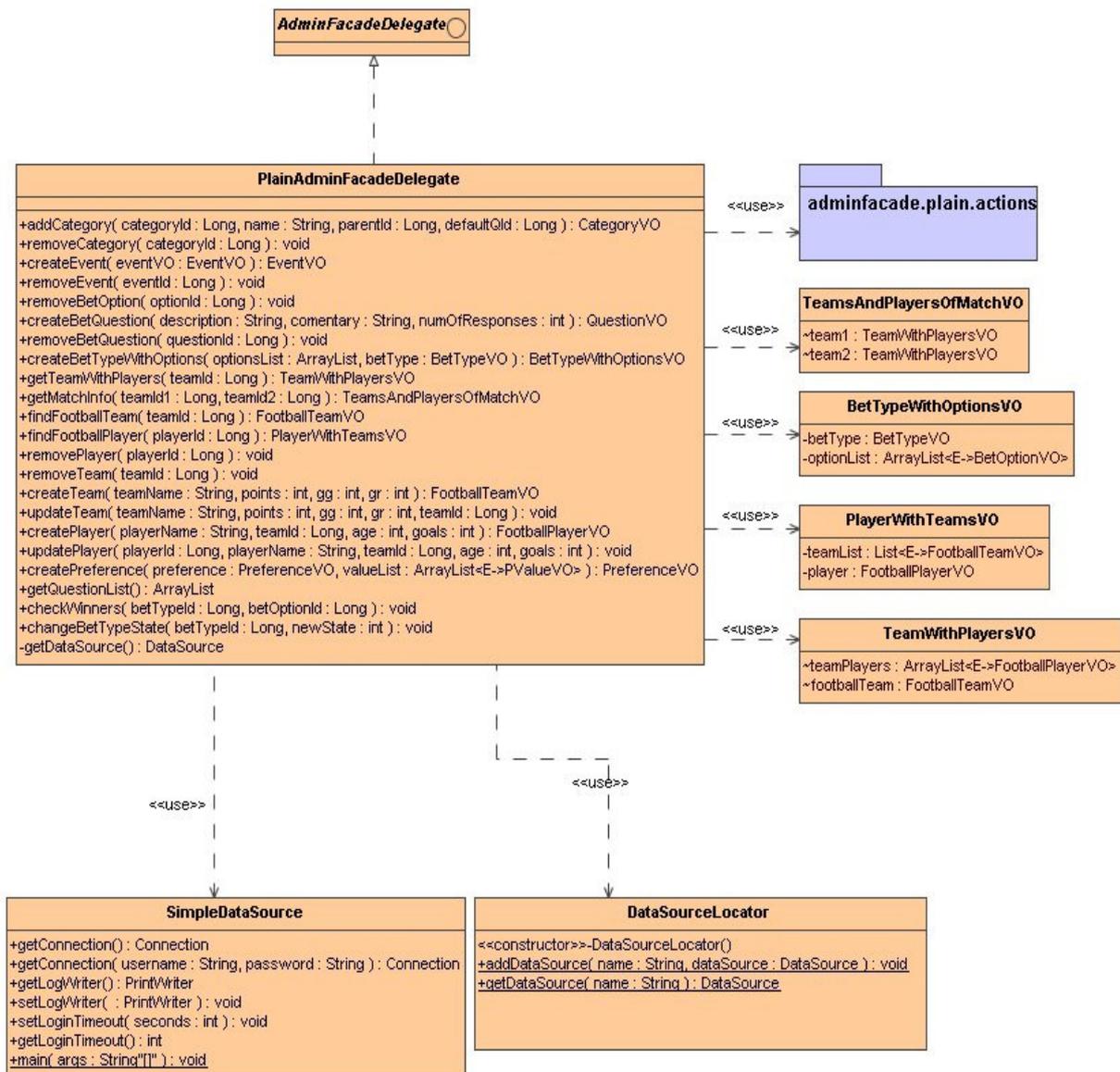
toAccountFromCreditCard: Su objetivo es realizar un ingreso en la cuenta del usuario desde una de las tarjetas bancarias registradas. Como parámetros de entrada tiene el dinero que se quiere ingresar y el número de la tarjeta desde donde se quiere realizar el ingreso.

transferMoney: Su objetivo es realizar una transferencia desde la cuenta de usuario hacia una cuenta bancaria. Como parámetros de entrada recibe el número de cuenta bancaria, el nombre y dirección del banco, el tipo de cuenta, el código swift, instrucciones adicionales y la cantidad que se quiere transferir.

updateUserProfileDetails: Su objetivo es actualizar la información personal de un usuario. Recibe un objeto `UserProfileDetailsVO` el cual encapsula los datos del usuario que se van actualizar.

En la fachada de usuario, además de hacer uso de las excepciones definidas para la toda la aplicación en el paquete `es.apuestasinparar.util`, se implementa una excepción que solo se utiliza en esta fachada y se encuentra en el paquete `es.apuestasinparar.model.userfacade.exceptions`. La clase es `IncorrectPasswordException` y se trata de una excepción que implementa `ModelException`. Se lanzará en aquellos casos donde se quiera indicar que dos contraseñas no coinciden, concretamente es en el método `login` y el método `changePassword`.

Las contraseñas de los usuarios se han cifrado con el fin de ofrecer en cierta medida privacidad y seguridad hacia las personas encargadas de la aplicación. Para ello se implementan las clases PasswordEncrypter y Jcrypt contenidas en el paquete es.apuestasinparar.model.userfacade.util y se utilizarán en las acciones que implementan los casos de uso de cambio de contraseña, autenticación y registro de usuarios.



• **Fachada de administrador**

Se trata de una fachada sin estado. Define las funcionalidades dirigidas únicamente al usuario administrador. A continuación se detallan los métodos de la fachada, especificando

su objetivo, sus parámetros de entrada y el valor o valores devueltos, detallando si es necesario el uso de un Custom Object.

addCategory: Su objetivo es añadir una nueva categoría al árbol de categorías y eventos. Como parámetros de entrada recibe el nombre de la categoría y el identificador de su categoría superior. Devuelve un objeto CategoryVO con identificador asignado en la creación.

changeBetTypeState: Su objetivo es cambiar el estado de un tipo de apuesta. Sus parámetros de entrada son el identificador del tipo de apuesta junto con el valor de su nuevo estado.

checkWinners: Su objetivo es determinar a los ganadores y perdedores de un tipo de apuesta, cambiando el estado de las apuestas, pagando a los ganadores y creando los movimientos financieros apropiados. Como parámetros de entrada recibe el identificador del tipo de apuesta y el identificador de la opción ganadora de ese tipo de apuesta.

createBetQuestions: Su objetivo es crear una nueva pregunta. Sus parámetros de entrada son la descripción de la pregunta, el comentario y el número de respuestas recomendado.

createBetTypeWithOptions: Su objetivo es crear un tipo de apuesta junto con sus opciones. Recibe como parámetros de entrada un objeto BetTypeVO con la información sobre el tipo de apuesta y una colección de opciones para el tipo de apuesta. Devuelve un custom object BetTypeWithOptionsVO. Este objeto contiene como atributo el objeto BetTypeVO que representa al nuevo tipo de apuesta creado y que contiene su identificador asignado, a su vez también tiene una colección de BetOptionVO con la información de opciones creadas para este tipo de apuesta, incluyendo su identificador asignado.

createEvent: Su objetivo es la creación de un nuevo evento. Sus parámetros de entrada es un objeto EventVO con los datos sobre el nuevo evento a crear. Devuelve un objeto EventVO con los mismos datos del objeto de entrada junto con su nuevo identificador asignado.

createPlayer: Su objetivo es la crear un nuevo jugador de fútbol. Recibe como parámetros de entrada el identificador del equipo al que pertenece, su edad y los goles

marcados. Devuelve un objeto `FootballPlayerVO` que encapsula la información del jugador junto con su identificador.

createPreference: Su objetivo es la crear una nueva preferencia junto con sus posibles valores. Recibe como parámetros un objeto `PreferenceVO` con la información sobre la nueva preferencia y una colección de objetos `PValueVO` con la información de cada valor posible para la preferencia.

createTeam: Su objetivo es crear un nuevo equipo de fútbol. Como parámetros de entrada tiene el nombre del equipo, sus puntos, los goles a favor y los goles en contra. Devuelve un objeto `FootballTeamVO` con identificador asignado.

findFootballPlayer: Su objetivo es recuperar la información de un jugador de fútbol. Como parámetro de entrada tiene el identificador del jugador buscado. Devuelve un objeto `PlayerWithTeamsVO` con los datos requeridos del jugador.

findFootballTeam: Su objetivo es recuperar la información de un equipo de fútbol. Como parámetro de entrada recibe el identificador del equipo solicitado. Devuelve un objeto `FootballTeamVO` con la información del equipo requerido.

getMatchInfo: Su objetivo es recuperar toda la información deportiva relativa a un partido de fútbol. Recibe como parámetros de entrada los dos identificadores de los equipos que participan en el partido. Devuelve un custom object `TeamsAndPlayersOfMatch`, el cual contiene como atributos dos custom objects `TeamWithPlayers`, cada uno de los cuales encapsula la información del equipo junto con una colección de sus jugadores.

getQuestionList: Su objetivo es recuperar la lista de preguntas existentes. Devuelve una colección de objetos `QuestionVO` con la información de cada una de las preguntas.

getTeamWithPlayers: Su objetivo es recupera la información relativa a un equipo y sus jugadores. Como parámetros de entrada recibe el identificador del equipo. Devuelve un custom object `TeamWithPlayers` que contiene la información del equipo y una colección de jugadores.

removeCategory: Su objetivo es eliminar una categoría. Recibe como parámetro de entrada el identificador de la categoría que se quiere borrar.

removeEvent: Su objetivo es eliminar un evento. Recibe como parámetro de entrada el identificador del evento.

removePlayer: Su objetivo es eliminar un jugador de fútbol. Recibe como parámetro de entrada el identificador del jugador.

removeTeam: Su objetivo es eliminar un equipo de fútbol. Recibe como parámetro de entrada el identificador del equipo que se quiere eliminar.

updatePlayer: Su objetivo es actualizar la información de un jugador de fútbol. Como parámetros de entrada recibe el nombre del jugador, el identificador del equipo al que pertenece, la edad y los goles.

updateTeam: Su objetivo es actualizar los datos de un equipo de fútbol. Sus parámetros de entrada son el nombre del equipo, sus puntos, los goles favor y los goles en contra.

• Fachada de apuestas

Se trata de una fachada con estado, dado por el talón de apuestas. Esta fachada proporciona funcionalidades específicas sobre la lógica de las apuestas. Engloba todo el tratamiento del talón del usuario y la creación de apuestas. Algunos casos de uso son exclusivos de usuarios autenticados y otros, relativos al talón de apuestas, están dirigidos a cualquier usuario. A continuación se detallan los métodos de la fachada, especificando su objetivo, sus parámetros de entrada y el valor o valores devueltos, detallando si es necesario el uso de un Custom Object.

createBetToTalon: Su objetivo es introducir una apuesta en el talón, esto es, introducir una opción elegida por el usuario para su talón de apuestas. Recibe como parámetros de entrada el identificador del tipo de apuesta, el identificador de la opción elegida, el identificador de la pregunta del tipo de apuesta y por último el identificador del evento.

createCombinedBet: Su objetivo es la creación de una apuesta combinada. Recibe como parámetros la cantidad de dinero y el login del usuario que realiza la apuesta. Devuelve un custom object CombinedBetWithNewBalance, el cual encapsula el nuevo

balance de la cuenta del usuario y la información de la apuesta realizada, necesaria para presentarla en el ticket de apuesta.

createSimpleBet: Su objetivo es la creación de una apuesta simple. Recibe como parámetros la cantidad de dinero, el login del usuario que realiza la apuesta y la posición de la opción escogida en el talón de apuestas. Devuelve un custom object `BetWithNewBalance`, el cual encapsula el nuevo balance de la cuenta del usuario y la información de la apuesta realizada, necesaria para presentarla en el ticket de apuesta.

getTalon: Su objetivo es devolver el talón de apuestas.

removeAllBetsFromTalon: Su objetivo es eliminar del talón todas las opciones previamente introducidas.

removeBetFromTalon: Su objetivo es eliminar una opción concreta del talón de apuestas. Recibe como parámetro la posición de la opción dentro del talón.

updateTalon: Su objetivo es actualizar la información de las opciones que contiene el talón de apuestas.

Fachada de catálogo de apuestas

El objetivo principal de esta fachada es definir las funcionalidades relativas al recorrido de categorías, eventos y tipos de apuestas. Los casos de uso que implementa no son concretos de ningún tipo de usuario. A continuación se detallan los métodos de la fachada, especificando su objetivo, sus parámetros de entrada y el valor o valores devueltos, detallando si es necesario el uso de un Custom Object.

findAllCategories: Su objetivo es recuperar la lista de categorías existentes. Devuelve una colección con objetos `CategoryVO` con la información de cada categoría.

findAllTeams: Su objetivo es recuperar la lista de equipos de fútbol. Devuelve una colección de objetos `FootballTeamVO` con la información de cada equipo.

findEvent: Su objetivo es buscar un evento. Tiene como parámetro de entrada el identificador del evento buscado. Devuelve un objeto `EventVO` con los datos del evento solicitado.

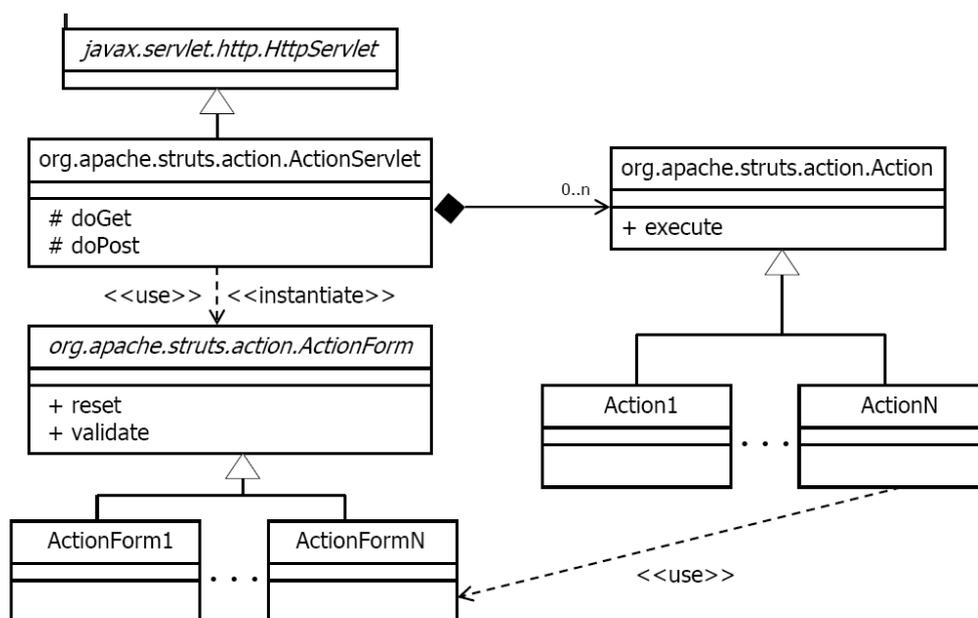
findLevelCategories: Su objetivo es recuperar la lista de categorías “hijas” de una misma categoría “padre”. Como parámetro de entrada recibe el identificador del “padre”. Devuelve una colección con las categorías solicitadas.

getCategoryWithEvents: Su objetivo es buscar los eventos junto con sus tipos de apuesta de un a categoría y por otro lado buscar los eventos de las subcategorías de la categoría anterior. Como parámetros de entrada recibe el identificador de la categoría padre, una lista de sus subcategorías y un valor boolean para indicar si se debe recuperar los tipos de apuestas cerrados o finalizados.

getEventWithBetType: Su objetivo es recuperar un evento junto sus tipos de apuesta. Como parámetros de entrada tiene el identificador del evento y un valor boolean para indicar si se deben recuperar los tipos de apuesta cerrados o finalizados.

7.2.4. Controlador

Alberga las acciones de la capa controlador conforme con el patrón arquitectónico Model View Controller. Estas acciones son la interfaz entre las peticiones en la capa vista y la capa de lógica de negocio. El usuario realiza las peticiones, el controlador se encarga de recibir la información de la capa vista y delegar en las operaciones de la lógica de negocio en el modelo. A continuación se detallan cada una las partes que conforman la capa controlador de nuestra aplicación.



12. Diagrama de clases de Apache Struts

• Acciones del controlador

Las acciones implementadas en el paquete `es.apuestasinparar.http.controller.actions` heredan de la clase `DefaultAction` del paquete `es.apuestasinparar.util.struts.action`. A su vez la clase `DefaultAction` hereda de la clase `Action` de Struts, sobrescribiendo el método `execute`. Todas las acciones están obligadas a implementar el método abstracto `doExecute`. Se está utilizando el patrón `Template Method`.

Cada una de las clases `action` del controlador hace uso de un solo método de una de las fachadas del modelo. En la llamada se le pasarán los datos necesarios para las operaciones en la lógica de negocio, y la capa modelo le devolverá los datos que el controlador requirió para la vista.

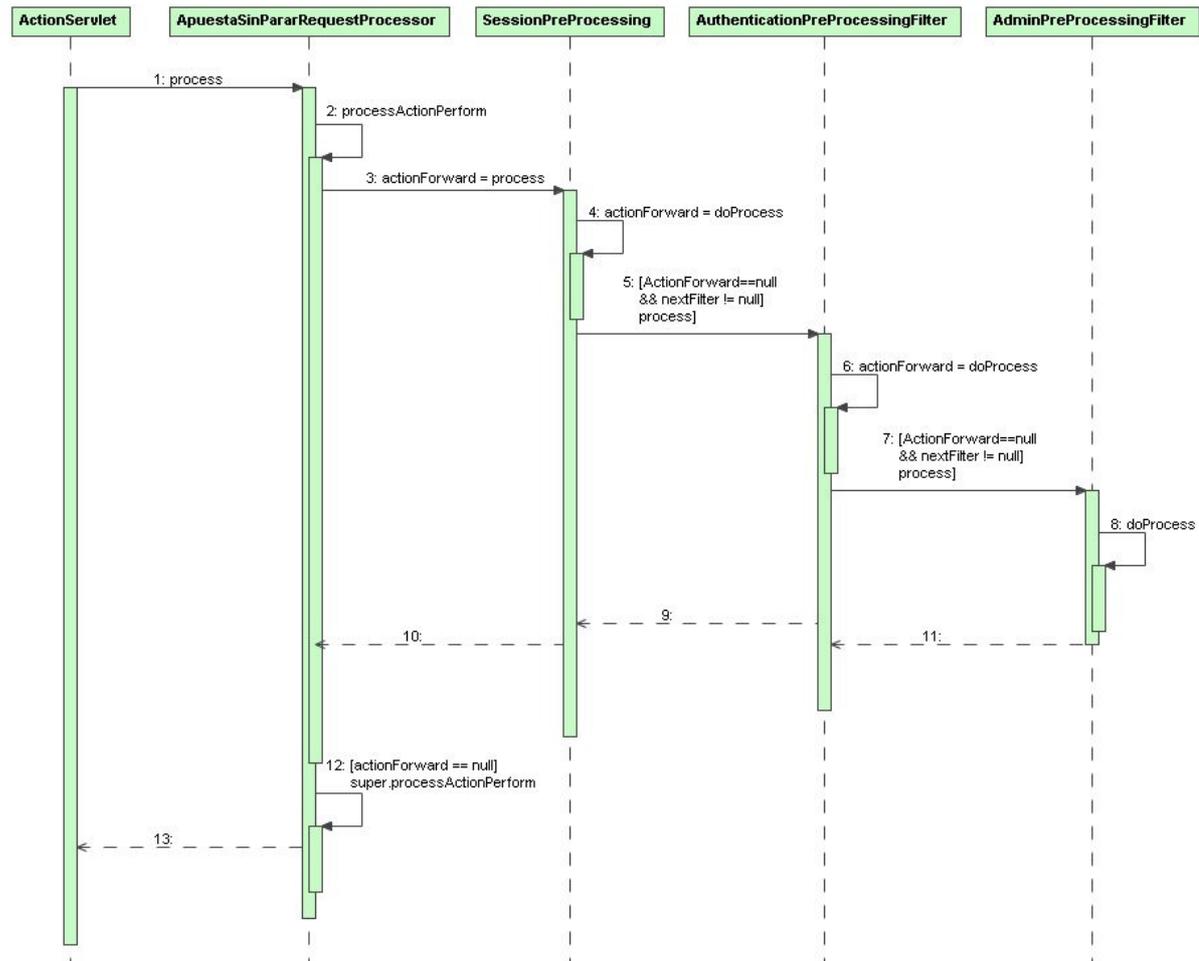
• Front Controller

Se añaden nuevas funciones al procesador de peticiones del framework Struts. Se define una clase que hereda de `TilesRequestProcessor` y una cadena de filtros fácilmente extensible. Mediante el atributo `firstPreProcessingFilter` se accede a dicha cadena. Las clases que engloban en el paquete `es.apuestasinparar.http.controller.frontcontroller` son las siguientes:

ApuestaSinPararRequestProcessor: Hereda la clase `TilesRequestProcessor`, definiendo el uso de un nuevo `RequestProcessor` en la aplicación. Como se comentó antes, mediante un `Chain Of Responsibility`, se crea una cadena de filtros a través de los cuales se descartarán o aceptarán peticiones, según restricciones tales como ser un usuario autenticado o ser administrador.

PreProcessingFilter: Clase abstracta que implementa la mencionada cadena de filtros de la aplicación. Se utiliza el patrón `Template Method`. Cada filtro debe implementar el método plantilla `doProcess`. Comenzando por el primer filtro, se recorrerán uno a uno hasta que no se cumpla uno y se devuelva un `ActionForward` definido para el caso, o si satisface todos los filtros, se pasa la petición a `TilesRequestProcessor`.

ApuestaSinPararMapping: Representa un `Java Bean` que hereda de `ActionMapping`, almacenando las restricciones impuestas a cada acción de Struts en el fichero de configuración de Struts.



13. Diagrama de secuencia del proceso que sigue una petición

• **Plugins**

Los plugins permiten registrar objetos en el ámbito de aplicación implementando la clase `PlugIn` del Framework Struts. En el paquete `es.apuestasinparar.http.controller.plugin` se implementa la clase `ApuestaSinPararPlugin`, la cual registrará una instancia de la clase `DateRanges` y otra de la clase `Catalogue`, que se encuentran en el paquete `es.apuestasinparar.http.view.applicationobjects`.

• **Sesión**

Se implementa una fachada que contendrá funcionalidades relativas a la gestión y control de la información que se mantiene en las sesiones de los usuarios, así como el manejo de cookies. Por tanto, cuando un caso de uso implique guardar o actualizar información en la sesión, o bien recuperar o enviar cookies, la acción del controlador trabajará contra la fachada de sesión y esta a su vez se encargará de llamar al método de la fachada del modelo, es decir, la fachada debe proporcionar una operación equivalente a la de la fachada de usuario por cada operación que necesite actualizar la sesión y/o cookies. Un ejemplo es el caso de uso de autenticación, donde se guardan datos del usuario en la sesión y en caso de ser seleccionada la opción de guardar contraseña, se enviarán cookies.

SessionManager
<pre> +FIRST_NAME_SESSION_ATTRIBUTE : String = "firstName"{readOnly} +USER_TYPE_SESSION_ATTRIBUTE : String = "userType"{readOnly} +LOGINNAME_SESSION_ATTRIBUTE : String = "loginName"{readOnly} +USER_BALANCE_SESSION_ATTRIBUTE : String = "balance"{readOnly} +PREFERENCES_SESSION_ATTRIBUTE : String = "preferences"{readOnly} +ADMIN_TYPE : byte = 1{readOnly} -USER_FACADE_DELEGATE_SESSION_ATTRIBUTE : String = "userFacadeDelegate"{readOnly} -BET_FACADE_DELEGATE_SESSION_ATTRIBUTE : String = "betFacadeDelegate"{readOnly} +LOGIN_NAME_COOKIE : String = "loginName"{readOnly} +ENCRYPTED_PASSWORD_COOKIE : String = "encryptedPassword"{readOnly} -COOKIES_TIME_TO_LIVE_REMEMBER_MY_PASSWORD : int = 30*24*3600{readOnly} -COOKIES_TIME_TO_LIVE_REMOVE : int = 0{readOnly} <<constructor>>-SessionManager() +login(request : HttpServletRequest, response : HttpServletResponse, loginName : String, clearPassword : String, rememberMyPassword : boolean) : void +registerUser(request : HttpServletRequest, loginName : String, clearPassword : String, userType : byte, userProfileDetailsVO : UserProfileDetailsVO) : void +updateUserProfileDetails(request : HttpServletRequest, userProfileDetailsVO : UserProfileDetailsVO, userType : byte) : void +changePassword(request : HttpServletRequest, response : HttpServletResponse, oldClearPassword : String, newClearPassword : String) : void +getCreditCard(request : HttpServletRequest) : ArrayList +logout(request : HttpServletRequest, response : HttpServletResponse) : void +getUserFacadeDelegate(request : HttpServletRequest) : UserFacadeDelegate +getBetFacadeDelegate(request : HttpServletRequest) : BetFacadeDelegate +touchSession(request : HttpServletRequest) : void +isUserAuthenticated(request : HttpServletRequest) : boolean +isAdminAuthenticated(request : HttpServletRequest) : boolean +updateSessionForAuthenticatedUser(request : HttpServletRequest, firstName : String, userType : byte, loginName : String, locale : Locale) : void +updateSessionFromCookies(request : HttpServletRequest) : void +leaveCookies(response : HttpServletResponse, loginName : String, encryptedPassword : String, timeToLive : int) : void +doLogin(request : HttpServletRequest, loginName : String, password : String, passwordsEncrypted : boolean, rememberMyPassword : boolean) : LoginResultVO +updateSessionBalance(request : HttpServletRequest) : void +updateUserPreferences(request : HttpServletRequest) : UserPWWithValuesVO </pre>

14. Diagrama de clase de la fachada SessionManager

7.2.5. Vista

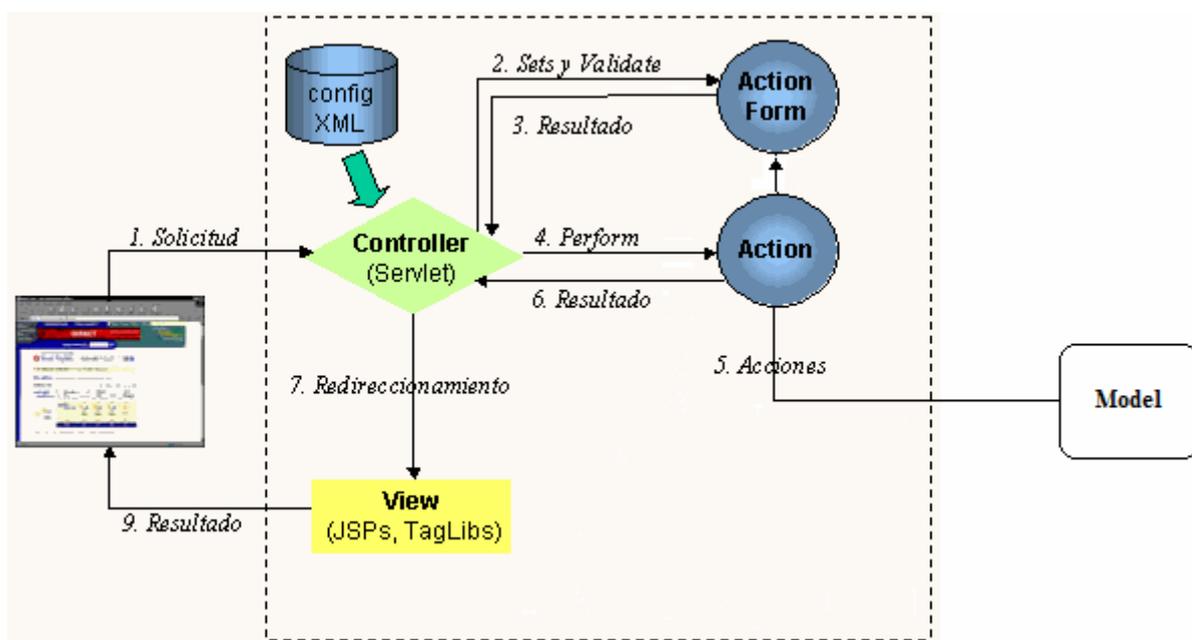
- **ActionForms**

Paquete que contiene los formularios que no se crean dinámicamente y que en su mayoría redefinen el método validator para realizar las comprobaciones más o menos complejas que se necesiten hacer sobre los datos

Los ActionForm Beans son clases que extienden ActionForm y que implementan métodos get y set para cada una de los inputs de un form de una página, además de los métodos validate y reset. Se debe escapar de realizar lógica de negocio en estos formularios.

Estos formularios ayudan a tratar los datos que se reciben en los formularios de una página, de forma que se pueden implementar el método validate para realizar comprobaciones más o menos complejas sobre los datos.

En la aplicación estos ActionForm se encuentra en el paquete es.apuestasinparar.http.view.actionforms.



15. Generación de una vista a partir de una petición

• Application Objects

Se implementan una serie de objetos que son accedidos en distintos ámbitos de la capa vista de la aplicación. En el paquete es.apuestasinparar.http.view.applicationobjectcs se encuentran las siguientes clases:

DateRanges: Proporciona una serie de métodos para obtener rangos de fechas, se hace uso de esta clase en algunas acciones del controlador donde se necesita introducir un día, mes y año en el ámbito de la request para poder se presentado en la JSP correspondiente.

Languages: Se trata de una clase que da soporte a la internacionalización de la aplicación proporcionando una relación entre el código y el nombre de los idiomas posibles en la aplicación.

Countries: Al igual que la clase Languages, da soporte a la internacionalización manteniendo una relación entre los códigos y nombres de los países.

Catalogue: Se trata de una clase cuyo objetivo es “cachear” el árbol de categorías. De esta forma evitamos el continuo acceso a base de datos cada vez que se soliciten las categorías.

- **Ficheros de internacionalización**

Ofrece el soporte para la internacionalización de la aplicación. Existirá un fichero por cada idioma que desea tener disponible y cada uno tendrá la lista de cadenas posibles en la aplicación traducidas al idioma correspondiente. Dependiendo del LOCALE la aplicación usará el fichero adecuado para mostrar el mensaje.

7.3. Subsistema 2

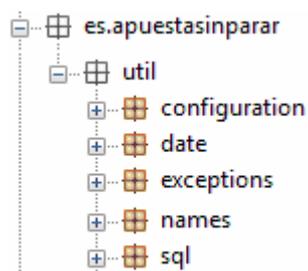
Este subsistema implementa una serie de clases de utilidad para el subsistema principal.

7.3.1. Objetivos

El objetivo es facilitar la implementación de diferentes capas del subsistema 1 y a su vez crear una estructura reutilizable en la medida de lo posible para otras aplicaciones.

7.3.2. Arquitectura

En la imagen se puede observar los subpaquetes de utilidad.



16. Arquitectura en paquetes del subsistema 2

7.3.3. Modelo

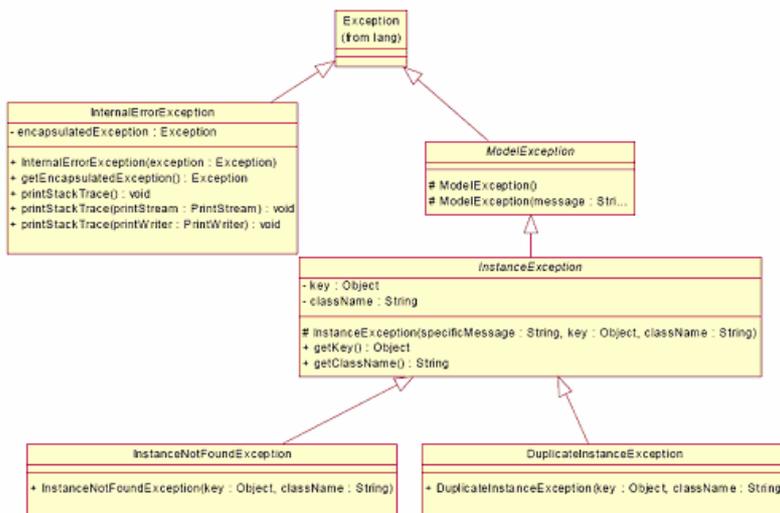
La clase `ConfigurationParametersManager` del paquete `util.configuration` es utilizada por el modelo para localizar parámetros de configuración, ya sea por un fichero `properties` o via `JNDI`.

En el paquete `util.exceptions` se implementan diferentes excepciones. La clase `InternalErrorException` representa un error “grave” en la ejecución de un método, permitiendo ocultar el API usada en la implementación del método. Esta excepción encapsula la excepción real y facilita la depuración. Por otro lado, las excepciones que heredan de `ModelException` representan errores lógicos en el modelo.

En el paquete `util.sql` se encuentra una serie de clases que dan soporte a la ejecución de las acciones del modelo, diferenciando las acciones transaccionales y las no transaccionales.

7.3.4. Controlador

El paquete `util.struts` contiene unas clases que facilitan la implementación de los formularios y las acciones del controlador usando `Struts`. Además también tiene la clase `PropertyValidator` que contiene una serie de métodos comunes de validación para los formularios.



17. Diagrama de clases del paquete de excepciones del subsistema 2

8. Implementación

8.1. Software requerido

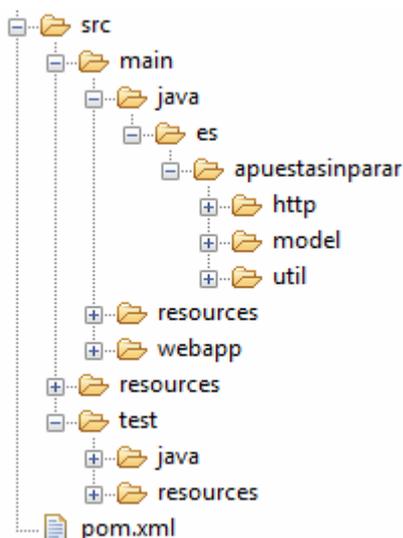
En primer lugar se debe tener instalado J2SE [9], que se proporciona en el disco que acompaña la memoria.

Para realizar la implementación del proyecto a partir de su distribución fuente se utilizará la herramienta de gestión de software Maven2. Como se explicó en capítulos anteriores, Maven2 gestiona de forma automática las dependencias del software, de forma que descargará los recursos necesarios de un repositorio remoto a un repositorio local.

En el disco que acompaña la memoria se encuentra una copia del repositorio (carpeta .m2_repo) ya preparado para la compilación de la aplicación. De todas formas con una conexión a Internet Maven2 realizará la descarga, pudiendo adquirir recursos actualizados.

8.2. Estructura

Para gestionar nuestra aplicación mediante Maven2 se ha seguido la estructura estándar que este define.



18. Estructura de la aplicación para su compilación con Maven 2

- /pom.xml: Fichero donde se definen los recursos necesarios.
- /src/main/java/: Contiene las fuentes java
- /src/test/java/: Contiene las pruebas de unidad siguiendo una estructura paralela al directorio /src/main/java.
- /src/main/resources/: Recursos Java Classpath que se necesiten
- /src/test/resources/: Recursos necesarios para las pruebas de unidad
- /src/main/webapp/: Raíz del webapp

8.3. Instrucciones de compilación

Primero se debe configurar una variable de entorno con el nombre JAVA_HOME y en valor, el directorio de instalación de nuestra distribución J2SE. Por otra banda, se necesita añadir a la variable de entorno PATH la ruta hacia la carpeta bin del directorio de instalación de Maven2. Una vez realizadas estas dos tareas, hay que dirigirse al directorio de la aplicación.

A continuación se comentan las instrucciones básicas de Maven2 para las diferentes opciones de compilación.

- Creación de War de la aplicación, ejecución automatizada de pruebas e instalación en el repositorio maven local de los componentes generados.

```
mvn install
```

- Generación de JavaDoc

```
mvn javadoc:javadoc
```

- Ejecución de scripts de base de datos

```
mvn sql:execute
```

- Generación de la distribución fuente de la aplicación en formato tar-gz y zip.

```
mvn assembly:assembly
```

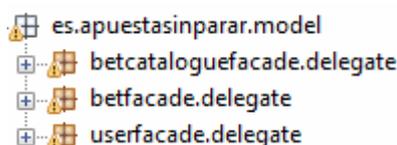
- Borrado de los directorios target del proyecto.

```
mvn clean
```

9. Pruebas

Se han implementado pruebas de unidad para las fachadas de usuario, la fachada de administrador, la fachada de catálogo y la fachada de apuestas. Durante la implementación de la aplicación, estas pruebas nos permiten probar las funcionalidades después de realizar cambios en el código.

Para las pruebas de la aplicación se ha creado un subsistema paralelo al subsistema principal. Su objetivo es proporcionar las clases necesarios para realizar las pruebas de unidad de las fachadas. Su arquitectura en paquetes es la siguiente.



19. Estructura en paquetes de las pruebas de unidad

• Introducción a las pruebas de unidad

Las pruebas de unidad se pueden definir cada una como una prueba individual de hardware o software, o grupos de unidades relacionadas.

- Normalmente, un test de unidad es un código que prueba una “unidad”: que puede ser una clase, un componente, un módulo o una función.
- Las pruebas se crean y se ejecutan mientras se desarrolla el software.
- Las pruebas de unidad no son pruebas funcionales de aplicación.
- Las pruebas de unidad no son interactivas.

Se ha utilizado JUnit [4] para realizar pruebas de unidad de la aplicación. Se trata de un intuitivo framework que nos permite realizar pruebas de unidad de forma automática. Proporciona una serie de aserciones que comprueban si los resultados son los esperados, es decir, los correctos. Las pruebas se pueden realizar de forma gráfica o en modo texto, en este caso las he realizado en modo texto.

Las pruebas deben ser independientes del repositorio de datos del que dispongamos, por tanto solo se utilizan métodos de fachadas y en ningún caso se utiliza directamente un DAO.

- **Modelo de pruebas**

La idea inicial era implementar una clase de prueba por fachada y probar sus métodos, pero debido a varios factores esto no se lleva a cabo con exactitud.

Por un lado, algunos casos de uso de una fachada se prueban en la clase de prueba creada para otra fachada. Esto ocurre por la lógica de la aplicación, por ejemplo, para realizar la prueba de la fachada de catálogo es necesario emplear muchos de los métodos de la fachada de administrador. En otros casos la causa es evitar la redundancia de código en dos clases de prueba, por ejemplo, si en la clase de pruebas para la fachada de apuestas se crean apuestas persistentes, se puede realizar la prueba para el caso de uso de ver el histórico de apuestas, perteneciente a la fachada de usuario.

Para dar soporte a las clases de prueba, se ha tenido que implementar una nueva fachada dedicada únicamente a esta labor. Básicamente proporciona métodos de borrado de datos y funciones de búsqueda de datos.

- **Implementación**

Los test implementados para cada fachada siguen el siguiente guión:

-Métodos AfterClass y BeforeClass donde se introducen en las tablas los datos necesarios para realizar correctamente todos los casos de prueba de la fachada, se inicializan las variables necesarias y en resumen, se prepara la clase para las pruebas. En el método BeforeClass se debe deshacer todos los cambios que posteriormente puedan influir en el funcionamiento de la aplicación, como por ejemplo, los datos que se introdujeron en las tablas. En resumen, la aplicación no debe notar la ejecución de las pruebas.

-Métodos After y Before, los cuales se ejecutan antes y después de cada prueba respectivamente. Los posibles cambios que influyen de una prueba a otra se restauran con estos métodos, de forma que cada prueba tenga lo necesario para su ejecución.

-Métodos Test, que son los casos de pruebas. Puede haber una prueba por caso de uso de la fachada, pero en ocasiones varios casos de uso se complementan entre sí, pudiendo englobarlos y probarlo en un solo caso de prueba. Cada prueba tendrá una o más aserciones que comprobarán los resultados. En otras pruebas, el propósito será comprobar que efectivamente se lance una excepción esperada.

A continuación se explicará brevemente las pruebas de la fachada de apuestas. Para dicha prueba de unidad, en el método BeforeClass se ha tenido que crear un árbol de categorías, eventos y tipos de apuesta con opciones. También se crea un usuario que realizará las apuestas. En el método AfterClass se borra toda esta información, en este punto ya se tuvo que implementar nuevos casos en la fachada test para poder borrar algunos datos. En los métodos Before y After que restauran algunos valores que se cambian en varias pruebas, como puede ser el talón de apuestas, que debe estar vacío antes de cada prueba, o el saldo de la cuenta del usuario. Por no detallar cada caso de prueba, se comenta uno ellos.

Caso de prueba de ganadores de apuesta

Se crean tres apuestas simples y una combinada. Se da por ganada una opción de un tipo de apuesta y se comprueba que efectivamente el usuario ha cobrado la apuesta que debe ganar, se crea el movimiento financiero adecuado y la apuesta pasa al estado perdida. A su vez tuvo que perder la otra apuesta simple del mismo tipo de apuesta con una opción no ganadora, se comprueba que el estado ha cambiado a perdida. La tercera apuesta simple debe seguir en espera. Respecto a la combinada, se comprueba que simplemente ha cambiado el estado de una de las hijas a ganada. Después se cancela otro tipo de apuesta y se comprueba que se ha cobrado el dinero de la tercera simple ganada, se genera el movimiento y cambió el estado a ganada. En la combinada se debe comprobar que el estado de la segunda hija ha pasado a ganada, por tanto el estado de la apuesta padre debe pasar a ganada y se mira si se cobra el dinero y se genera el movimiento.

10. Conclusiones y Futuras Líneas de Trabajo

• Análisis del trabajo realizado

La realización de este proyecto tenía varios objetivos. En primer lugar se ha creado de forma satisfactoria una aplicación web con la plataforma J2EE y con una arquitectura guiada por el patrón arquitectónico MVC. Por otra parte se han cumplido todos los objetivos marcados en cuanto a requisitos funcionales, y en el tiempo marcado en la planificación, no obstante con algunas prisas finales.

Otros objetivos particulares eran aprender el conjunto de tecnologías J2EE necesarias para el desarrollo de aplicaciones Web, utilizar frameworks OpenSource tales como Apache Struts, el aprendizaje de patrones de diseño para el desarrollo de una aplicación con arquitectura MVC y por último realizar un enfoque de desarrollo iterativo. Analizando dichos objetivos, se puede establecer que se han cumplido en su totalidad. En un principio mis conocimientos sobre muchas de las tecnologías utilizadas en el desarrollo de la aplicación eran casi nulos, sin embargo ahora puedo decir que he adquirido unos conocimientos que me permiten realizar una aplicación web con un respetable nivel de calidad. Por otro lado he aprendido a utilizar diferentes patrones de diseño que permiten realizar un buen diseño de aplicaciones, no solo de este tipo.

A pesar de lo aprendido, soy consciente de que estos conocimientos son una base que me servirá para profundizar mucho más en futuro.

• Líneas futuras

A continuación se comentan mejoras para incorporar en un futuro a la aplicación. Gracias al diseño realizado, la aplicación podrá adquirir nuevas características y funcionalidades sin un gran coste.

Optimización de transacciones con la base de datos: En varias acciones del modelo se hacen llamadas a métodos de un DAO varias veces, como por ejemplo, en un bucle. Sería conveniente cambiar esta situación para que estas llamadas se resumieran en una sola.

Seguridad en la transmisión de datos: Se trata de una mejora bastante importante debido al trato de cuentas bancarias, tarjetas de crédito y otros temas delicados en seguridad. Un usuario debe tener plena confianza en el trato de su dinero y demás temas relacionados. Para la transmisión de estos datos sensibles se suele usar SSL (Secure Sockets Layer) y el protocolo HTTPS (HTTP over SSL). El uso de SSL debe limitarse a los momentos en que se vayan transmitir los datos que queremos proteger, esto es debido a que la encriptación que realiza SSL ralentiza de forma notable la tasa de datos. Una extensión llamada `sslex` permite usar SSL con Struts, ayudando a resolver muchos de los problemas de conmutación HTTP/HTTPS.

Mejora de la interfaz gráfica: Sin duda uno de los atractivos de una aplicación web es su interfaz, un usuario que se sienta incomodo visualmente hablando no se sentirá cómodo en ningún momento, por tanto, una mejora visual facilitaría la satisfacción del cliente.

Información y eventos específicos: Ahora mismo solo existe información de la primera división de la liga española y sólo existen eventos específicos para dicha liga. La aplicación está preparada para incorporar nuevos tipos de eventos específicos de forma inmediata. Dicha incorporación facilitará el trabajo del administrador así como el acceso a mayor información deportiva por parte del usuario cuando esté observando el tipo de apuesta.

Cuotas de apuestas variables: Las cuotas de las opciones de apuesta se calculan a partir de la probabilidad de la opción en el momento de la creación del tipo de apuesta junto con sus opciones. Una mejora sería que esta probabilidad variara en el tiempo en base al número de usuarios que hayan elegido dicha opción, de forma que las opciones más escogidas verán aumentada su probabilidad y viceversa con las menos elegidas:

11. APÉNDICE

11.1. Instalación del Software / Manual de Usuario

La aplicación ha sido probada en los sistemas operativos Windows Vista y Windows XP. En cuanto a software, se ha utilizado J2SE 5.0 update 11, Maven2, MySQL Community Edition 5.0.27, mysql-connector-java-5.0.4-bin.jar (driver JDBC 3), servidor de aplicaciones Apache tomcat, JSTL 1.1.2, y Apache Struts 1.2.9.

MySQL

Instalación: Para instalar MySQL en los sistemas operativos Windows Vista y Windows XP, ejecutar el fichero Setup.exe situado en la carpeta \resources\MySQL\mysql-5.0.27-win32 del disco, y en el proceso de instalación, escoger la instalación típica. Una vez instalado, marcar la opción para configurar MySQL.

Configuración Elegir configuración detallada y en las siguientes pantallas escoger las siguientes opciones:

Server Type: Server machine.

Database usage: Transactional database only

Number of concurrent connections: Online transaction processing

Configuración de opciones de Windows: Include bin directory in Windows path.

Si en la pantalla de puesta en funcionamiento de la configuración aparece un error, intentar darle a retry y ver si se soluciona en error, en otro caso, seguir instrucciones de la pantalla, pues puede ser que un antivirus o firewall impida la puesta en marcha

Ahora hay que crear las tablas necesarias para la aplicación, para facilitar la creación, se usará una interfaz gráfica. Para ello ejecutar el fichero mysql-gui-tools-5.0-r6-

win32.msi situado en la carpeta \resources\MySQL\mysql-gui-tools y seguir las instrucciones de instalación. Al finalizar ir al menú inicio, mysql y pinchar en mysqladministrator. En Server Host poner localhost, en username poner root y en password la contraseña introducida en la instalación. En el menú de la izquierda ir a 'Catalogs', y abajo pinchar con el botón derecho y 'Create New Schema', como nombre poner 'afz'. Ahora vamos crear un usuario con privilegios para el manejo de esta base de datos, para ello pinchar en el menú de la izquierda en 'User Administration', 'New User' y poner como user 'afz' y como password 'afz'. Una vez creado el usuario, ir a la pestaña 'Schema Privileges'. Pinchar en el Schema 'afz' y aplicarle todos los privilegios. Ya tenemos configurada nuestra base de datos.

J2SE

Para instalar J2SE, ejecutar el fichero jdk-1_5_0_11-windows-i586-p.exe situado en la carpeta \resources\Java\J2SE y seguir el proceso normal de instalación.

Apache Tomcat

Para instalar Apache Tomcat en los sistemas operativos Windows Vista y Windows XP, ejecutar el fichero apache-tomcat-5.5.20.exe situado en \resources\ApacheTomcat y seguir el proceso de instalación. Una vez instalado hay que realizar algunas configuraciones:

En al directorio de instalación, copiar el driver JDBC (mysql-connector-java-5.0.4-bin.jar) a /common/lib.

Definir un datasource global de nombre "jdbc/afz": Para ello, añadir las siguientes líneas a \$APACHE_HOME/conf/server.xml, dentro de la etiqueta <GlobalNaminResources>

```
<Resource name="jdbc/PFC"
    auth="Container"
    type="javax.sql.DataSource"
    driverClassName="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost/afz"
    username="afz"
    password="afz"
    maxActive="4"
    maxIdle="2"
    maxWait="10000"
    removeAbandoned="true"
```

```
removeAbandonedTimeout="60"  
logAbandoned="true"  
validationQuery="SELECT COUNT(*) FROM PingTable" />
```

Añadir las siguientes líneas a \$APACHE_HOME/conf/context.xml, dentro de la etiqueta <Context>

```
<ResourceLink name="jdbc/PFC" global="jdbc/PFC"  
type="javax.sql.DataSource"/>
```

Llegado aquí ya se puede proceder al despliegue de la aplicación. Para ello dirigirse al menú inicio, programas y entrar en Apache Tomcat 5.5. Luego ejecutar Monitor Tomcat y en el icono que aparecerá en la barra de inicio a la derecha, botón derecho del ratón y start service. A continuación dirigirse al explorador, teclear localhost:8080 en la barra de direcciones, entrar y en el menú de la izquierda pinchar en 'Tomcat Manager'. Introducir el nombre y contraseña puesta en la instalación de Tomcat. En la siguiente pantalla, abajo se puede examinar ficheros y desplegar el .war de la aplicación. Luego pinchar en la aplicación que aparecerá entre las Aplicaciones de la pantalla manager.

Un paseo por la aplicación

La página de inicio contiene el formulario para identificarse y la opción para un registrarse un usuario nuevo. En la parte izquierda de la pantalla está el menú de deportes para poder acceder a los diferentes eventos y apuestas. También está la opción para ver acceder a la información deportiva de liga española de fútbol.



20. Página principal de apuestasinparar.es

A continuación se puede ver el menú de un usuario ya identificado cuando se pincha en el enlace “Mi cuenta”. En este caso se trata de un usuario administrador, por lo que también se puede acceder al menú de administrador pinchando en “Menú de administrador”.



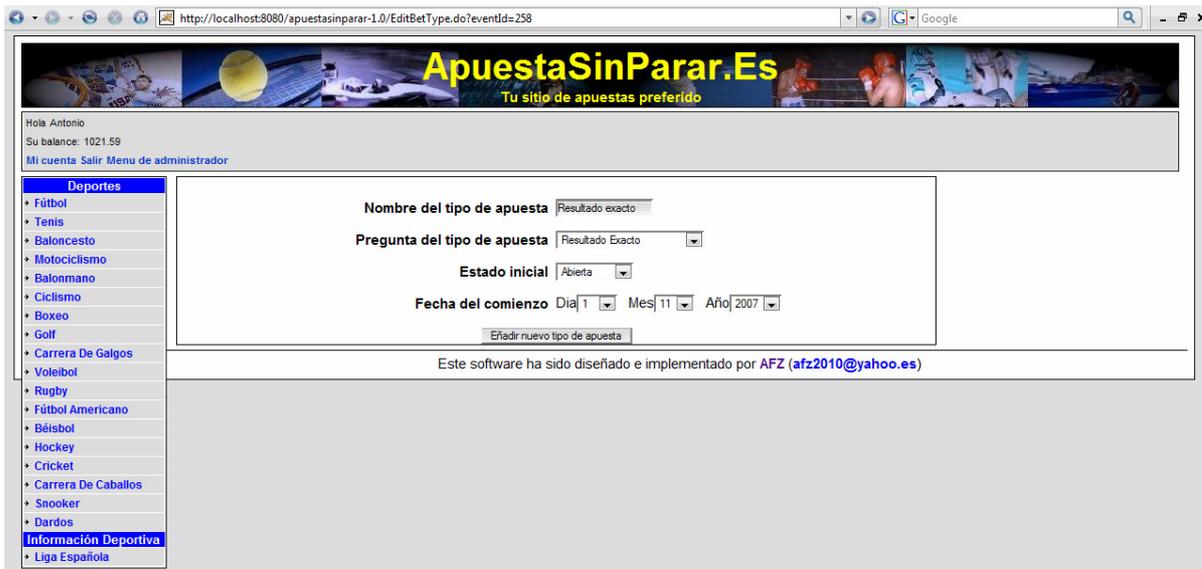
21. Menú de la cuenta de usuario de apuestasinparar.es

A continuación se puede observar a un usuario administrador añadiendo un nuevo evento específico de partido de fútbol de la primera división de la liga española, se puede ver como se introduce el título del evento igual que un evento genérico, y luego selecciona los dos equipos participantes, de forma que este evento estará relacionado con estos dos equipos para luego poder mostrar información de estos si el usuario la solicita.



22. Creación de un evento específico de fútbol por parte del administrador

Ahora el administrador va proceder a la creación de un tipo de apuesta sobre el evento anteriormente creado. Para ello se le pone un nombre, se elige el tipo de pregunta de la apuesta que en este caso es “resultado exacto”, como estado se le pone “Abierta” y la fecha a partir de la cual automáticamente la apuesta se cierra hacia el usuario.



23. Creación de un tipo de apuesta por parte del administrador

A continuación se puede ver como el usuario va realizar una apuesta simple, para ello primero introduce en su talón de apuestas las opciones que le interesa.



24. Usuario introduciendo opciones de apuesta en su talón de apuestas

Ya ha introducido en su talón de apuestas dos opciones, a continuación accede al talón e introduce la cantidad a apostar en una opción, por ejemplo, la de Rafa Nadal. Pulsa Apostar y ya está creada la apuesta.



25. Usuario administrando su talón de apuestas

11.2. Contenido del CD

La estructura de directorios del disco que acompaña a la memoria es la siguiente

\bin: Distribución binaria.

\bin\webapps: Fichero WAR de la aplicación.

\bin\resources\sql: Scripts de creación de tablas.

\docs\api: Documentación JavaDoc.

\docs: Archivo PDF de esta memoria.

\m2_repo: Copia del repositorio de Maven2

\resources: Utilidades y tecnologías necesarias

\src: Distribución fuente en un fichero tar.gz y un fichero .zip.

A continuación se nombran las aplicaciones incluidas en el directorio resources del disco, son OpenSource y compatibles tanto con Windows XP como con Windows Vista.

J2SE: Implementación de J2SE 1.5 update11.

Maven2: Herramienta software para la gestión del ciclo de vida de un proyecto.

MySQL Community Edition 5.0.27: Está acompañado de una aplicación para la gestión la base de datos con una atractiva interfaz gráfica.

Apache Tomcat 5.5.20: Servidor de aplicaciones web J2EE con soporte para Servlet 2.4 y JSP 2.0.

JSTL 1.1.2: Librería de tags estándar.

JUNIT: Framework para la creación de las pruebas de unidad.

Mysql-connector-java-5.0.4: Driver JDBC3.

Las versiones son compatibles con Windows Vista y Windows XP.

12. Bibliografía

[1] Apache Maven Project

<http://maven.apache.org/>

[2] Keogh, Jim. *J2EE manual de referenrecia*

[3] The Apache Software Foundation

<http://www.apache.org>

[4] JUnit, Testing Resources for Extreme Programming

<http://www.junit.org/>

[5] Bellas, Fernando. *Apuntes de integración de sistemas*

[6] Ted Husted, Cedric Dumoulin, George F., David W. *Struts In Action. Manning Greenwich 2003*

[7] Hans Bergsten. *JavaServer Pages, 2nd Edition. Prentice Hall 2003*

[8] Java 2 Platform Enterprise Edition, v 1.4. API Specification

<http://java.sun.com/j2ee/1.4/docs/api/>

[9] Java 2 Platform Standard Edition 5.0. API Specification

<http://java.sun.com/j2se/1.5.0/docs/api/>

[10] XHTML Tutorial

<http://www.w3schools.com/xhtml>

[11] Apache Struts

<http://struts.apache.org/>

[12] The J2EE™ 1.4 Tutorial

<http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html>

[13] JavaServer Pages Standard Tag Library 1.1 Tag Reference
<http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/index.html>

[14] CSS Tutorial
<http://html.conclase.net/tutorial/html/5/2>

[15] Patrones de Diseño en aplicaciones Web con Java J2EE
http://java.ciberaula.com/articulo/disenio_patrones_j2ee/

12.1. Enlaces de Interés

http://java.ciberaula.com/articulo/disenio_patrones_j2ee/

<http://www.tic.udc.es/~mad/teaching/pfc3/>

<http://cvs.peopleware.be/training/maven/maven2/>

http://www.programacion.com/java/tutorial/joa_struts/3/

http://cricava.com/java/unit_testing_ii_aprendiendo_a_usar_junit

<http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/index.html>

<http://www.1x4x9.info/files/jstl/html/online-chunked/ar01s03.html#N1083A>

13. Glosario

Clase: Definición de un tipo de objetos que tienen unas características comunes. Una clase es un patrón para crear nuevos objetos. Contiene atributos y métodos.

Caso de uso: Especificación de las secuencias de acciones, incluyendo secuencias, variantes y secuencias de error, que pueden ser efectuadas por un sistema, subsistema o clase por iteración con autores externos.

Autenticar: Demostrar que su identidad es realmente la que dice ser.

Driver: También llamado controlador, se encarga de actuar como interfaz entre dos sistemas para permitir la comunicación entre ellos.

Escalabilidad: Medida de capacidad de un sistema para mejorar su rendimiento incorporando nuevos elementos de procesamiento sobre los que ejecutarse sin la necesidad de modificar su implementación para conseguir este fin.

Aplicación Web: Se trata de un software que los usuarios utilizan desde un servidor web a través de la red. Últimamente están obteniendo una creciente popularidad gracias a la habilidad para actualizarlas y mantenerlas sin distribuir software en miles de potenciales clientes.

Framework: Es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Un framework representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

Interfaz: Dispositivo de comunicación entre dos sistemas; conexión física y funcional entre dos entidades independientes.

Objeto: Es una entidad que tiene un estado y un conjunto definido de operaciones (métodos) que operan sobre este estado.

OpenSource: Término que hace referencia a un tipo de software del cual está disponible el código fuente y cuyos términos para su distribución han de cumplir: redistribución libre, inclusión de código fuente y las modificaciones pueden ser redistribuidas bajo condiciones de Copyleft.

Paquete: Término que denota un mecanismo de propósito general para organizar un grupo de elementos.

Patrón arquitectónico: Patrón de alto nivel que propone una arquitectura global para una aplicación

Patrón de diseño: Solución a un problema de diseño no trivial que es efectiva y reusable.

Request: Cualquiera de las ordenes o peticiones que se envían a un servidor por medio de una aplicación cliente.

Requisito funcional: Requisito que especifica una acción que debe ser capaz de realizar el sistema, sin considerar restricciones físicas.

Requisito no funcional: Requisito que especifica propiedades del sistema, como restricciones del entorno o de implementación, rendimiento, dependencias de la plataforma, mantenibilidad, extensibilidad o fiabilidad.

Servlet: Clase Java que recibe peticiones, normalmente http y genera una salida, que puede ser HTML, WML o XML.

Subsistema: Componente importante de un sistema organizado de una forma coherente. Un sistema puede ser dividido en subsistemas usando particiones o niveles.

API: Un API (del inglés Application Programming Interface - Interfaz de Programación de Aplicaciones, interfaz de programación de la aplicación) es un conjunto de especificaciones de comunicación entre componentes software. Representa un método para conseguir abstracción en la programación, generalmente (aunque no necesariamente) entre los niveles o capas inferiores y los superiores del software.

14. Acrónimos

J2EE: Java 2 Enterprise Edition.

J2SE: Java Standard Edition.

JSP: Java Server Pages.

JSTL: JSP Standard Tag Library.

JNDI: Interfaz de Nombrado y Directorio Java.

XML: eXtensible Markup Language.

XHTML: eXtensible Hypertext Markup Language.

CSS: Cascading Style Sheets.

API: Application Programming Interface.

DAO: Data Access Object.

TO: Transfer Object.

