J2EE: Eurokilo

Autor: SERGIO BEAUMONT CRUZ
I.T.I.S.

Consultor: OSCAR ESCUDERO SÁNCHEZ

18 jun. 2007

INDICE

1.- RESUMEN.

El inicio de este proyecto vino potenciado por una afición común entre unos amigos y el autor de esta aplicación. La idea principal era la de facilitar la gestión de una peña de quinielas de fútbol, así como agilizar la tarea de envío para los participantes de ésta.

El primer paso fue el de entrar en materia teniendo en cuenta el escaso conocimiento de la tecnología ofrecida para implementar esta aplicación. Una vez empecé a informarme sobre el J2EE, poco a poco, pude ver el gran potencial de este tipo de programación. Por eso, a pesar de que la bibliografía es muy escasa en mis consultas, es cierto que he encontrado mucha información por la red y mucha variedad de contenidos al respecto.

Una vez en materia, intenté olvidarme del objetivo del proyecto para centrarme en las posibilidades de esta tecnología. La idea era, una vez alcanzado este punto, tener la mente más abierta para disponer de más opciones a la hora de implementar mi proyecto inicial y por esta misma razón, asigné más del 30% del tiempo en investigar y encontrar las herramientas más adecuadas y, como no, a llegar a entender lo mejor posible su funcionamiento. De este modo, y tras leer varias versiones sobre esta tecnología, poco a poco fui decantándome por el uso de un servidor JBOSS (a decir verdad, fue el primero que vi y el que me convenció), de una base de datos Hypersonic (integrada en el servidor de aplicaciones), de la tecnología Hibernate (con su lenguaje HSQL, casi idéntico al conocido SQL) y de la herramienta XDoclet para la creación de los documentos XML necesarios para la comunicación de todas las partes del entorno.

Por último, antes de iniciar con el proyecto, decidí intentar utilizar todas las herramientas elegidas sin ayuda de ningún IDE (posteriormente me decidiría por el eclipse, ya que he llegado a conocerlo bastante bien). Siguiendo con la investigación, instalé todos los módulos necesarios para las pruebas iniciales (**JAVA** (jdk 1.5.0.11), **JBOSS** (GA-4.0.5), **ANT** (1.7.0) y **XDOCLET** (1.2.3)).

Conseguido el objetivo de conocer bien las herramientas, comencé a utilizar el IDE y a emplear los conocimientos obtenidos para el verdadero problema: mi proyecto. Así que, tras pasar por varias versiones de mi aplicación, llegué al resultado que expondré en los siguientes puntos.

2.- INTRODUCCION.

2.1.- Justificación del proyecto.

Las tecnologías disponibles hoy en día son tan avanzadas como para gestionar casi cualquier situación que se pueda dar en la vida cotidiana. En mi caso, el hecho de buscar entretenimiento en una afición que compartimos muchos, me llevó a agilizar un poco más el proceso de gestión y disfrutar más de nuestro hobby. Por esta razón comenzó este proyecto, que como era de esperar, no fue la única. El hecho de conocer este tipo de herramientas y utilizarlas para algo tan esencial como nuestro entretenimiento, lo hacía mucho más interesante.

El punto de partida era "casi" nulo, ya que nunca antes había trabajado con este tipo de tecnología (J2EE), sin embargo, disponía de ciertos conocimientos sobre servidores Web (Apache, IIS), HTML, SQL y Java que dieron más agilidad al aprendizaje. Esto no hizo más que animarme a conocer estas herramientas.

2.2.- Objetivos del proyecto.

Generales

El objetivo principal del proyecto es el de aprender el uso de estas tecnologías, hasta ahora desconocidas por mí, y conseguir que cualquier persona con un simple navegador Web, sea capaz de ver el resultado y sacar provecho del mismo. Para ello, he pensado en una Web diseñada para gestionar un peña de quinielas, en la que cada usuario puede decidir el signo de cada casilla. Digamos que esta peña tiene algo distinto a las demás, y es que la quiniela la deciden los propios usuarios y no el administrador de la peña.

Específicos

Los usuarios que utilizarán la aplicación son de dos tipos, los administradores y los participantes.

Los administradores se encargarán de gestionar las quinielas (publicarla, cerrarla, establecer dobles, premios, etc.), las temporadas (añadir jornadas, partidos, publicar resultados, etc.), los usuarios, las cuentas de los usuarios, etc. También podrán disponer de la aplicación como un participante más, es decir, podrán enviar su propia quiniela para calcular la jugada.

Por otro lado, los participantes podrán enviar sus propias quinielas para que la aplicación pueda calcular la jugada que se realizará. El resto de servicios para los participantes es de simple información sobre las clasificaciones, los ingresos, los gastos, los resultados de los partidos y de las quinielas, etc.

2.3.- Metodología.

Para la realización del proyecto, he optado por una división en fases. Empezando por el análisis de la aplicación describiendo los casos de uso y el análisis de la interfaz de usuario. Siguiendo con el diseño de la aplicación, en el que, según el modelo de tres capas, esquematizo el diagrama de clases, diseño un primer formato para la interfaz gráfica del usuario y realizo un primer diseño de la base de datos con sus tablas y relaciones. Y terminando con la implementación de la aplicación, en la que me dedico expresamente al código para alcanzar los diseños establecidos anteriormente.

El diseño de la base de datos ira cambiando según las necesidades del proyecto, ya que mientras voy avanzando, van apareciendo nuevos problemas que solucionar con los datos representados.

2.4.- Planificación del proyecto.

Según las fechas propuestas al inicio del proyecto por el plan docente, dividí las tareas en estas secciones. Aunque las dos ultimas fases (Análisis y Diseño) las mezclé un poco ya que aún no tenía claro el diseño de la base de datos y según el mismo, podría optar por un mapa Web o por otro, según las opciones que quería barajar, por lo que intenté realizar un primer diseño de la base de datos y observar la forma más optima de almacenar los datos para luego tomar mi decisión.

Plan de trabajo (PEC 1)

- Primer acercamiento con el entorno
- Decisión de los objetivos a alcanzar
- División de la carga del trabajo

Análisis del proyecto (PEC 2)

- Instalación de las herramientas necesarias
- Estudio de los casos de uso
- Realización del diagrama de secuencias
- Mapa Web
- Instalación del entorno

Diseño del proyecto (PEC 3)

- Diseño de la base de datos (persistencia)
- Diseño de la interfaz de usuario (presentación)
- Diseño de entidades (negocio)

2.5.- Producto obtenido

El producto obtenido es una aplicación Web destinada al entretenimiento.

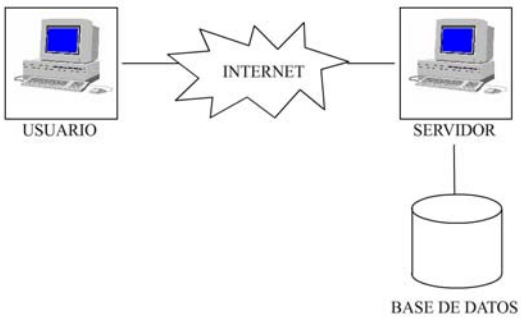
La tecnología utilizada es bastante fácil de instalar en cualquier servidor que cuente con una conexión a Internet, bastará con instalar el servidor de aplicaciones JBoss y añadir el archivo EAR a la carpeta DEPLOY del mismo. Actualmente "casi" todos los equipos tienen un paquete java, por lo que seguramente no será necesario instalarlo, aunque tal vez sea una buena opción la actualización del mismo. A parte de la instalación del servidor y la actualización del java, podría ser interesante una versión del ANT, para trabajar directamente con la base de datos, ya que se trata de la base de datos integrada en el JBOSS (Hypersonic), y su actualización es más cómoda a través de un XML y, por supuesto, de su ejecución mediante la herramienta mencionada (ANT).

La administración de la aplicación puede realizarse vía Web, por lo que no será necesario que el servidor cuente con un navegador ya que la administración se puede realizar desde cualquier cliente Web que cuente con un navegador. Sin embargo, las futuras actualizaciones hacen mucho más rentable contar con un IDE del tipo Eclipse instalado en el servidor a fin de facilitar los futuros cambios en el código.

El archivo EAR que contiene la aplicación, no es más que un paquete que cuenta con páginas JSP y hojas de estilo CSS, además de todas las clases java compiladas y los XML necesarios para su ejecución. Más adelante expondré un esquema con las clases utilizadas en la aplicación y el mapa de las páginas utilizadas.

3.- Análisis de requisitos de la aplicación.

La aplicación será un sistema por el cuál se pueda tanto gestionar los datos de una peña de quinielas, como utilizar el sistema para recoger información y realizar acciones tales como enviar una quiniela de un usuario concreto. Las conexiones se establecerán siempre a través de Internet. Se accederá al servidor de aplicaciones que a su vez accederá al servidor de base de datos que se encuentra en el mismo equipo.



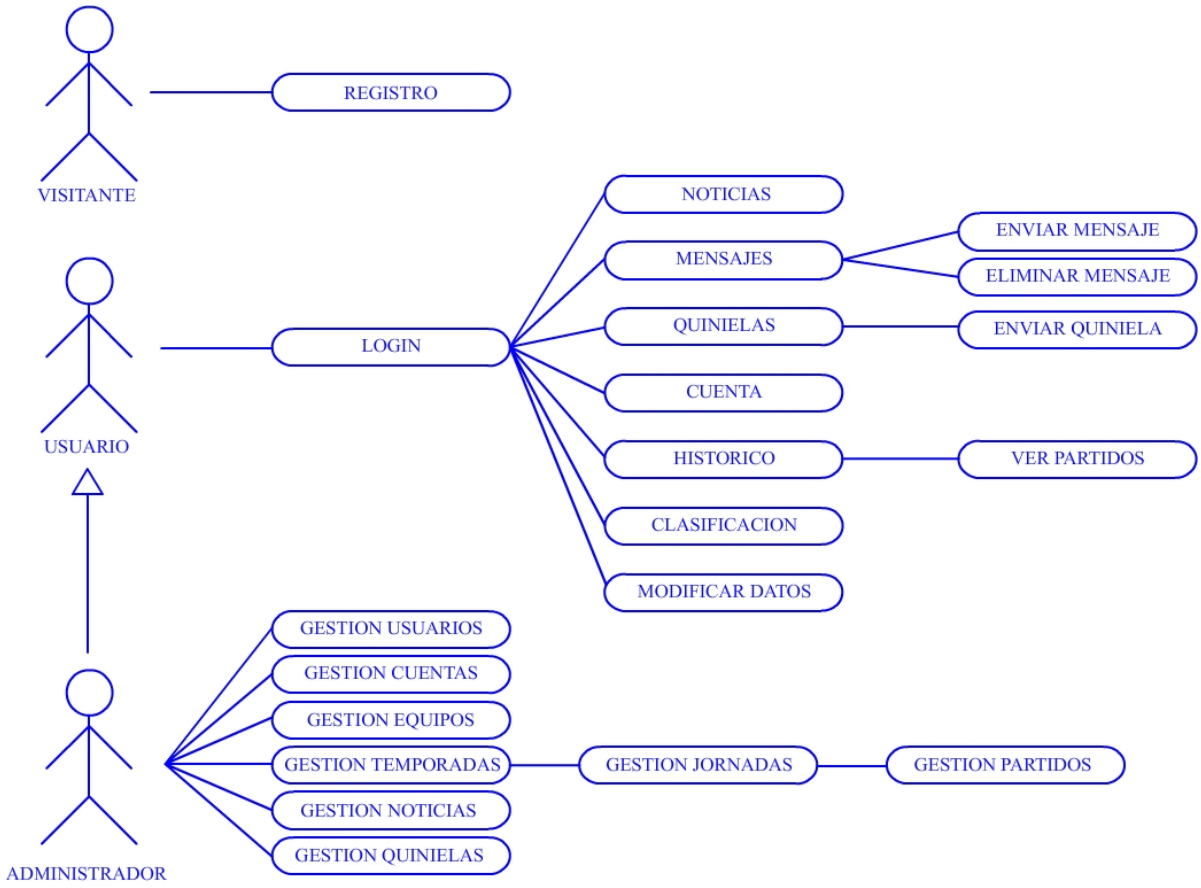
3.1.- Identificación de los actores.

- Visitante: Los visitantes solo tienen un posible papel, el de registro. Desde el que pondrán sus datos y un mensaje de texto para, posteriormente, ser enviado al administrador (o a los administradores).

- Usuario: Los usuarios tienen varios papeles. Quizá el más importante sea el de enviar quiniela, desde el que se almacena la jugada del usuario en la base de datos. A parte de este papel, disponen de otras posibles acciones como la de ver la clasificación, ver los resultados de los partidos, ver el detalle de sus cuentas, etc.

- Administrador: Por último, los administradores tienen todos los papeles de gestión de la aplicación. Gestión de quinielas, gestión de temporadas, gestión de jornadas, gestión de equipos, gestión de usuarios y gestión de cuentas. Desde estos papeles, tendrán opción a mantener actualizada la base de datos con los eventos que se vayan sucediendo (premios en quinielas, partidos jugados, nuevos usuarios, etc.)

3.2.- Esquema gráfico de los casos de uso.



3.3.- Documentación textual de los casos de uso.

REGISTRO.
Resumen de funcionalidad: Se envía un mensaje a los administradores con los datos del invitado que quiere registrarse.
Actores: Invitado.
Precondición: Ninguna.
Poscondición: Los datos del invitado quedan almacenados como mensaje en el buzón de los administradores.
Descripción: El Invitado debe introducir sus datos (Nombre, Login, Correo) y un mensaje opcional en el formulario. Una vez envíe estos datos, irán al buzón de los administradores, los cuales se pondrán en contacto con él para posteriormente darle de alta como usuario si lo estiman oportuno.

LOGIN.
Resumen de funcionalidad: Los usuarios se identifican en el sistema.
Actores: Usuario y Administrador.
Precondición: Los usuarios deben estar registrados.
Poscondición: Los usuarios acceden al sistema.
Descripción: El usuario registrado debe introducir su login y su password correctamente para poder entrar al sistema y tener accesibles sus opciones de gestión (en el caso del administrador) o de información (en el caso del usuario).

NOTICIAS.
Resumen de funcionalidad: Se muestran las noticias de la peña de quinielas.
Actores: Usuario y Administrador.
Precondición: El usuario debe estar registrado.
Poscondición: El usuario accede a la vista de todas las noticias.
Descripción: Se muestra por pantalla la lista de noticias total, ordenada por fechas.

MENSAJES.
Resumen de funcionalidad: Se muestran los mensajes recibidos y enviados.
Actores: Usuario y Administrador
Precondición: El usuario debe estar registrado. Los mensajes deben pertenecer al propio usuario, este puede estar tanto en el campo origen como en el destino.
Poscondición: El usuario accede a su buzón de mensajes.
Descripción: Se muestran por pantalla tanto los mensajes recibidos por el usuario como los enviados por el mismo.

ENVIAR MENSAJES.
Resumen de funcionalidad: Se crea un mensaje con un origen y un destino.
Actores: Usuario y Administrador
Precondición: El usuario debe estar registrado.

Poscondición: El mensaje creado se envía al buzón del usuario destino y se marca como "no leído".
Descripción: El usuario accede a un formulario en el que debe especificar el nombre del usuario al que va dirigido, un título de mensaje y un texto. Una vez guarde los datos, estos pasan a la base de datos y se marcan con el campo leído a "0" (estado: No leído) y el propio usuario como origen del mensaje.

ELIMINAR MENSAJES.
Resumen de funcionalidad: Se elimina el mensaje del buzón del usuario.
Actores: Usuario y Administrador
Precondición: El usuario debe estar registrado. El mensaje que se va a eliminar debe tener al propio usuario como destino.
Poscondición: El mensaje queda eliminado totalmente de la base de datos.
Descripción: Se elimina el mensaje del buzón del usuario al que va dirigido y de la lista de mensajes enviados del que lo envía (campo origen).

QUINIELAS.
Resumen de funcionalidad: Se muestran las quinielas del usuario y de la peña.
Actores: Usuario y Administrador
Precondición: El usuario debe estar registrado.
Poscondición: Se muestran las quinielas del usuario y de la peña con sus datos (aciertos, gastos, premios).
Descripción: Se muestra por pantalla la lista de todas las quinielas jugadas agrupadas por temporada. El usuario puede ver el detalle de cada una de ellas.

ENVIAR QUINIELA.
Resumen de funcionalidad: El usuario envía su quiniela para la semana actual.
Actores: Usuario y Administrador
Precondición: El usuario debe estar registrado. Debe haber alguna quiniela activa. El usuario que intente enviar la quiniela, debe tener suficiente dinero en su cuenta.
Poscondición: La quiniela del usuario queda almacenada en la base de datos.
Descripción: El usuario debe seleccionar el resultado (1, X o 2) para cada uno de los partidos de la quiniela activa. Una vez envíe el resultado, quedará almacenado en la base de datos y posteriormente, tendrá opción a modificarla hasta que esta quiniela esté cerrada.

CUENTA.
Resumen de funcionalidad: El usuario obtiene el detalle de los movimientos de su cuenta.
Actores: Usuario y Administrador
Precondición: El usuario debe estar registrado. El usuario debe ser el propietario de la cuenta.
Poscondición: Se muestran todos los movimientos de la cuenta desde el inicio.

Descripción: Se accede a la lista de movimientos de la cuenta del usuario ordenada por fechas. Se muestran los ingresos, los gastos y el total de la cuenta en cada momento.

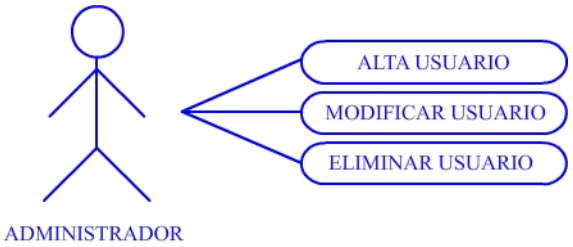
HISTÓRICO.
Resumen de funcionalidad: El usuario puede navegar entre las distintas temporadas y jornadas.
Actores: Usuario y Administrador
Precondición: El usuario debe estar registrado.
Poscondición: Se selecciona una temporada y una jornada.
Descripción: Se muestra una lista con las temporadas almacenadas en la base de datos y, por cada temporada, la lista de jornadas disponibles para la misma.

VER PARTIDOS.
Resumen de funcionalidad: Se muestran los partidos de la jornada seleccionada.
Actores: Usuario y Administrador
Precondición: El usuario debe estar registrado.
Poscondición: Se muestran los partidos con sus resultados (si se han jugado) de la temporada y jornada seleccionadas.
Descripción: Se muestra una lista con los partidos de la jornada seleccionada. Si los partidos se han jugado, se muestra también el resultado de los mismos.

CLASIFICACIÓN.
Resumen de funcionalidad: El usuario puede ver la clasificación de la temporada actual.
Actores: Usuario y Administrador
Precondición: El usuario debe estar registrado. Debe haber como mínimo un partido jugado.
Poscondición: Se muestra la clasificación actual.
Descripción: Se muestra la clasificación de los equipos agrupados por división. Los datos mostrados son: Partidos jugados (PJ), Goles a favor (GF), Goles en contra (GC) y puntos (PTS). La lista se ordena por número de puntos y por goles a favor.

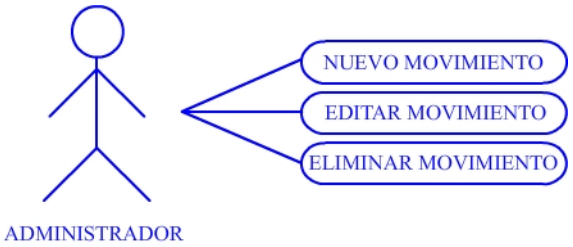
MODIFICAR DATOS.
Resumen de funcionalidad: El usuario puede modificar sus datos de usuario.
Actores: Usuario y Administrador
Precondición: El usuario debe estar registrado.
Poscondición: Los datos del usuario quedan modificados.
Descripción: El usuario accede a un formulario en el que se reflejan sus datos actuales. Tiene opción a cambiar su nombre completo y su contraseña. En este último caso, deberá repetirla dos veces como medida de seguridad. El resto de los datos, el login y el correo, no podrá modificarlo por sí mismo, y deberá enviar un mensaje a los administradores para solicitar el cambio.

GESTIÓN DE USUARIOS.



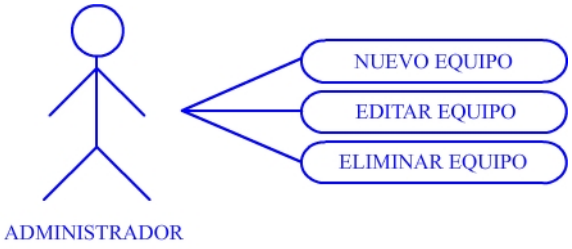
Resumen de funcionalidad: El usuario puede dar de alta, modificar y eliminar usuarios.
Actores: Administrador
Precondición: El usuario debe estar registrado como Administrador. En el caso de las altas y las modificaciones, el login del nuevo usuario no puede repetirse. En el caso de las modificaciones y las eliminaciones, el usuario debe existir en la base de datos.
Poscondición: Según la acción, el usuario queda almacenado en la base datos (altas), puede también quedar actualizado (modificación) o puede quedar eliminado (eliminación).
Descripción: El administrador accede a un panel de control en el que puede ver la lista de todos los usuarios dados de alta. Tendrá opciones para dar de alta un nuevo usuario, modificar un usuario existente o eliminar a un usuario ya registrado.

GESTIÓN CUENTAS.



Resumen de funcionalidad: El usuario puede añadir, modificar o eliminar movimientos en las cuentas.
Actores: Administrador
Precondición: El usuario debe estar registrado como Administrador. En el caso de las modificaciones y bajas, el movimiento de cuenta en cuestión, debe existir en el sistema. En el caso de las altas, la cantidad debe ser superior a cero.
Poscondición: Se crea un movimiento nuevo (alta). Puede también modificarse un movimiento existente (modificación) o puede eliminarse un movimiento existente (baja).
Descripción: Se muestra una lista con el estado de las cuentas de cada usuario. Desde esa lista, se puede acceder al detalle de cada cuenta y en ese momento, el administrador podrá añadir nuevos movimientos a la cuenta del usuario seleccionado, modificar un movimiento existente, o eliminar un movimiento de la base de datos.

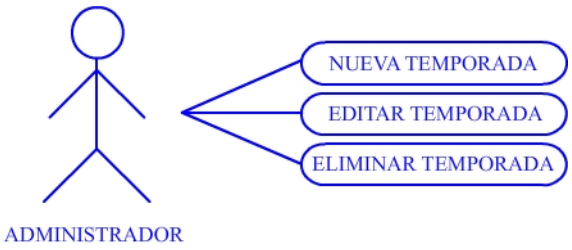
GESTIÓN DE EQUIPOS.



Resumen de funcionalidad: El usuario puede dar de alta, modificar y eliminar equipos.
Actores: Administrador
Precondición: El usuario debe estar registrado como Administrador. En el

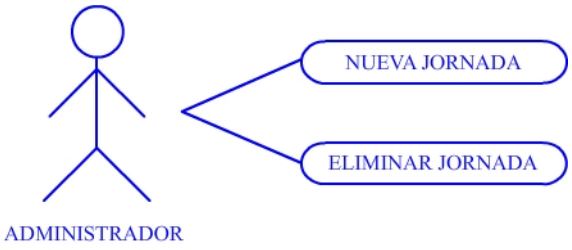
caso de las modificaciones y las eliminaciones, el equipo debe existir en la base de datos. En el caso especial de las bajas, el equipo no puede estar en ningún partido almacenado.
Poscondición: Según la acción, el equipo queda almacenado en la base datos (altas), puede también quedar actualizado (modificación) o puede quedar eliminado (eliminación).
Descripción: El administrador accede a una lista de todos los equipos almacenados en el sistema. Desde esa lista podrá añadir nuevos equipos, rellenando un simple formulario con el nombre del equipo. Podrá también modificar el nombre de un equipo concreto o eliminar un equipo existente.

GESTIÓN DE TEMPORADAS.



Resumen de funcionalidad: El usuario puede dar de alta, modificar y eliminar temporadas.
Actores: Administrador
Precondición: El usuario debe estar registrado como Administrador. En el caso de las modificaciones y las eliminaciones, la temporada debe existir en la base de datos. En especial, en el caso de las bajas, no pueden haber jornadas en la temporada que se va a eliminar.
Poscondición: Según la acción, la temporada queda almacenada en la base datos (altas), puede también quedar actualizada (modificación) o puede quedar eliminada (baja).
Descripción: En el caso de las altas o modificaciones, el administrador accede a un formulario en el que tendrá que especificar una descripción. En el caso de las bajas, la temporada queda eliminada del sistema.

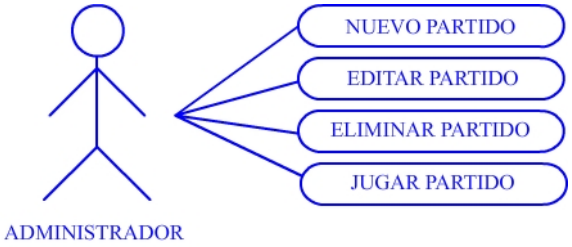
GESTIÓN DE JORNADAS.



Resumen de funcionalidad: El usuario puede dar de alta y eliminar jornadas.
Actores: Administrador
Precondición: El usuario debe estar registrado como Administrador. En el caso de las eliminaciones, la jornada debe existir en la base de datos y no pueden haber partidos en la jornada.
Poscondición: Según la acción, la jornada queda almacenada en la base datos (altas) o puede quedar eliminada (baja).
Descripción: En el caso de las altas, se añade una nueva jornada con el número inmediatamente superior al último generado. En el caso de las bajas, la jornada queda eliminada del sistema.

GESTIÓN DE PARTIDOS.

En este caso, veo importante explicar por separado los casos de uso, así que los procesos de alta, edición y baja, los explicaré como he hecho en los anteriores casos de uso, mientras que el caso "Jugar Partido" lo explicaré a parte. A continuación muestro con detalle el caso de uso GESTIÓN DE PARTIDOS.



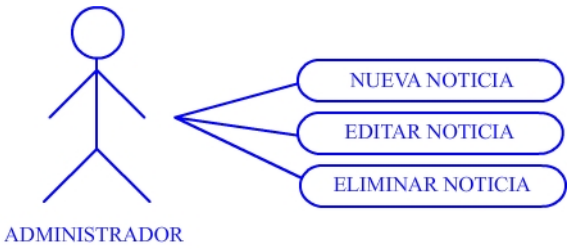
*** NUEVO PARTIDO, EDITAR PARTIDO, ELIMINAR PARTIDO.**

Resumen de funcionalidad: El usuario puede dar de alta, modificar y eliminar partidos.
Actores: Administrador
Precondición: El usuario debe estar registrado como Administrador. En el caso de las ediciones y bajas, el partido debe existir en la base de datos y no pueden haberse jugado.
Poscondición: Según la acción, el partido queda almacenado en la base datos (altas), puede quedar actualizado (edición) o puede quedar eliminado (baja).
Descripción: En el caso de las altas y las ediciones, se accede a un formulario en el que se debe indicar el equipo local y el equipo visitante. En el caso de las bajas, el partido queda eliminado de la base de datos.

*** JUGAR PARTIDO.**

Resumen de funcionalidad: El usuario establece un resultado para el partido.
Actores: Administrador
Precondición: El usuario debe estar registrado como Administrador. El partido debe existir en la base de datos
Poscondición: El partido pasa a estado "jugado" y se almacena su resultado en la base de datos. Se actualizan los aciertos de las quinielas.
Descripción: El usuario accede a un formulario en el que se debe indicar el numero de goles del equipo local y el del equipo visitante. Una vez se guarda el resultado, se comprueba que existan quinielas que contengan ese partido. Si existen, se actualiza el número de aciertos de la peña para esa quiniela y el de cada una de las quinielas de los usuarios que han jugado esa semana. También se modifica automáticamente la clasificación, en caso de tratarse de la temporada actual.

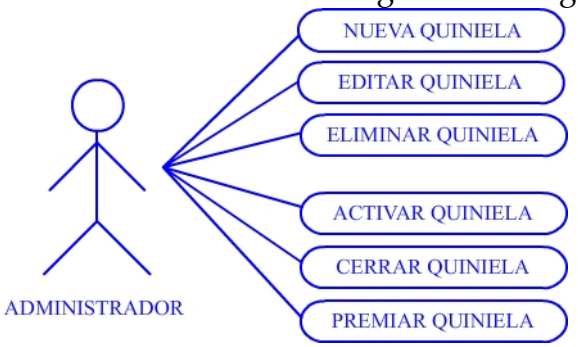
GESTIÓN DE NOTICIAS.



Resumen de funcionalidad: El usuario puede dar de alta, modificar y eliminar noticias.
Actores: Administrador
Precondición: El usuario debe estar registrado como Administrador. En el caso de las ediciones y las bajas, la noticia debe existir en la base de datos.
Poscondición: Según la acción, la noticia queda almacenada en la base datos (altas), puede quedar actualizada (edición) o puede quedar eliminada (baja).
Descripción: En el caso de las altas, se accede a un formulario en el que se debe escribir el texto de la noticia. Una vez guardada, se almacena en la base de datos con la fecha actual.

GESTIÓN DE QUINIELAS.

Al igual que el caso de los partidos, dividiré este caso de uso en varios casos. Por un lado, explicaré los procesos de alta, edición y baja. Por otro lado, explicaré otros tres casos por separado: Activación, Cierre y Premiar. Cada uno de estos tiene su propia definición, por lo que mostraré el caso de uso individual por cada uno de ellos. A continuación muestro con detalle el Caso de Uso gráfico de la gestión de quinielas:



*** NUEVA QUINIELA, EDITAR QUINIELA, ELIMINAR QUINIELA.**

Resumen de funcionalidad: El usuario puede dar de alta, modificar y eliminar quinielas.
Actores: Administrador
Precondición: El usuario debe estar registrado como Administrador. En el caso de las ediciones, la quiniela debe existir en la base de datos. En el caso de las bajas, además de existir la quiniela, no puede tener partidos asignados.
Poscondición: Según la acción, la quiniela queda almacenada en la base datos (altas), puede quedar actualizada (edición) o puede quedar eliminada (baja).
Descripción: En el caso de las altas, se accede a un panel de control en el que debemos especificar el número de dobles y la cantidad jugada. Si no se especifica la cantidad (se deja a cero), la aplicación la calculará automáticamente por el número de dobles. Una vez almacenada la quiniela, se dará la posibilidad al usuario de asignarle partidos a la quiniela. Estos

partidos deben haber sido dados de alta previamente. El administrador también tiene opción a modificar la quiniela, ya sea cambiando el número de dobles o el gasto de la misma, o modificando los partidos que formarán parte de la quiniela en cuestión (ya sea cambiando un partido por otro, o eliminando dicho partido de la quiniela). En el caso de dar de baja una quiniela, tenemos una precondición de que la quiniela no contenga partidos jugados. Esta definición conlleva otro caso de uso para gestionar los partidos de una quiniela, pero estaría repitiendo el mismo esquema en el caso de uso de los partidos.

*** ACTIVAR QUINIELA.**

Resumen de funcionalidad: El usuario activa la quiniela.
Actores: Administrador
Precondición: El usuario debe estar registrado como Administrador. La quiniela debe existir en el sistema.
Poscondición: La quiniela queda en estado "Activa".
Descripción: El usuario, tras realizar esta acción, deja la quiniela con estado "Activa" para que los usuarios puedan acceder a la jugada correspondiente a esta quiniela.

*** CERRAR QUINIELA.**

Resumen de funcionalidad: El usuario cierra la quiniela activa.
Actores: Administrador
Precondición: El usuario debe estar registrado como Administrador. La quiniela debe existir en el sistema. Debe haber alguna quiniela activa.
Poscondición: La quiniela queda en estado "Cerrada" y las cuentas de los usuarios se actualiza.
Descripción: El usuario, tras realizar esta acción, deja la quiniela con estado "Cerrada". En ese momento, la aplicación comprueba los usuarios que participaron en la quiniela (aquellos que tenían suficiente saldo en sus cuentas) y calcula la cantidad que jugará cada uno de ellos a partir del gasto total de la quiniela (según su número de dobles o la cantidad establecida por el administrador). Una vez calculada esta cantidad, se añade un nuevo movimiento a las cuentas de los usuarios de carácter negativo, restando la cantidad correspondiente a la cuenta de cada usuario que participe esa semana.

*** PREMIAR QUINIELA.**

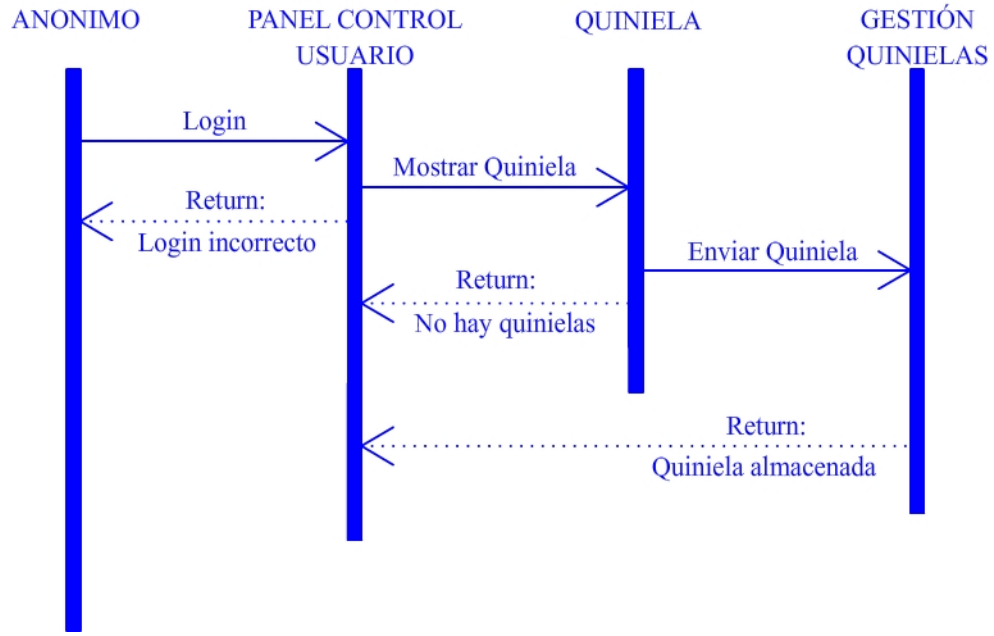
Resumen de funcionalidad: El usuario establece un premio para la quiniela seleccionada.
Actores: Administrador
Precondición: El usuario debe estar registrado como Administrador. La quiniela debe existir en el sistema..
Poscondición: La quiniela queda actualizada con el premio establecido.

Descripción: El usuario accede a un formulario en el que establece una cantidad que se corresponde con el premio ganado de la quiniela seleccionada. En ese momento, se comprueba el número de participantes de la misma, y tras calcular el premio a repartir entre cada uno de ellos, actualiza las cuentas de los usuarios, añadiendo un nuevo movimiento con la cantidad positiva y su descripción correspondiente.

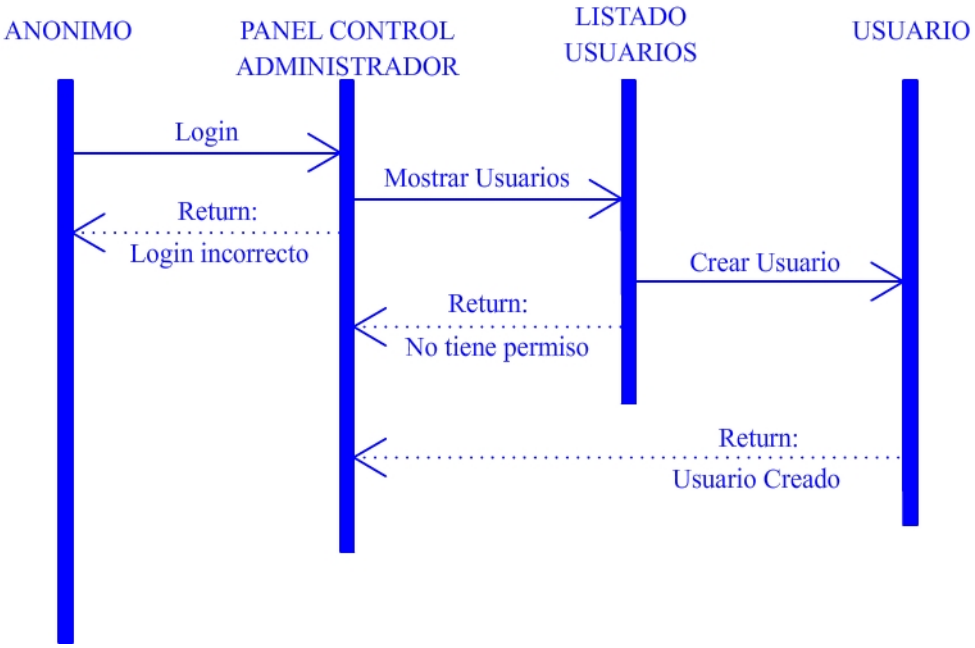
3.4.- Diagrama de secuencias.

A continuación muestro las operaciones más importantes con su diagrama de secuencias. En el caso de los usuarios, sólo muestro la operación de envío de quinielas, y en el caso de los administradores, muestro operaciones como la de gestionar usuarios. El resto de operaciones del administrador son muy parecidas a ésta, por lo que no las detallaré. En cuanto al resto de operaciones de los usuarios, son todas de consulta a la base de datos, y conociendo el esquema de tres capas, se hace muy lógico su detalle.

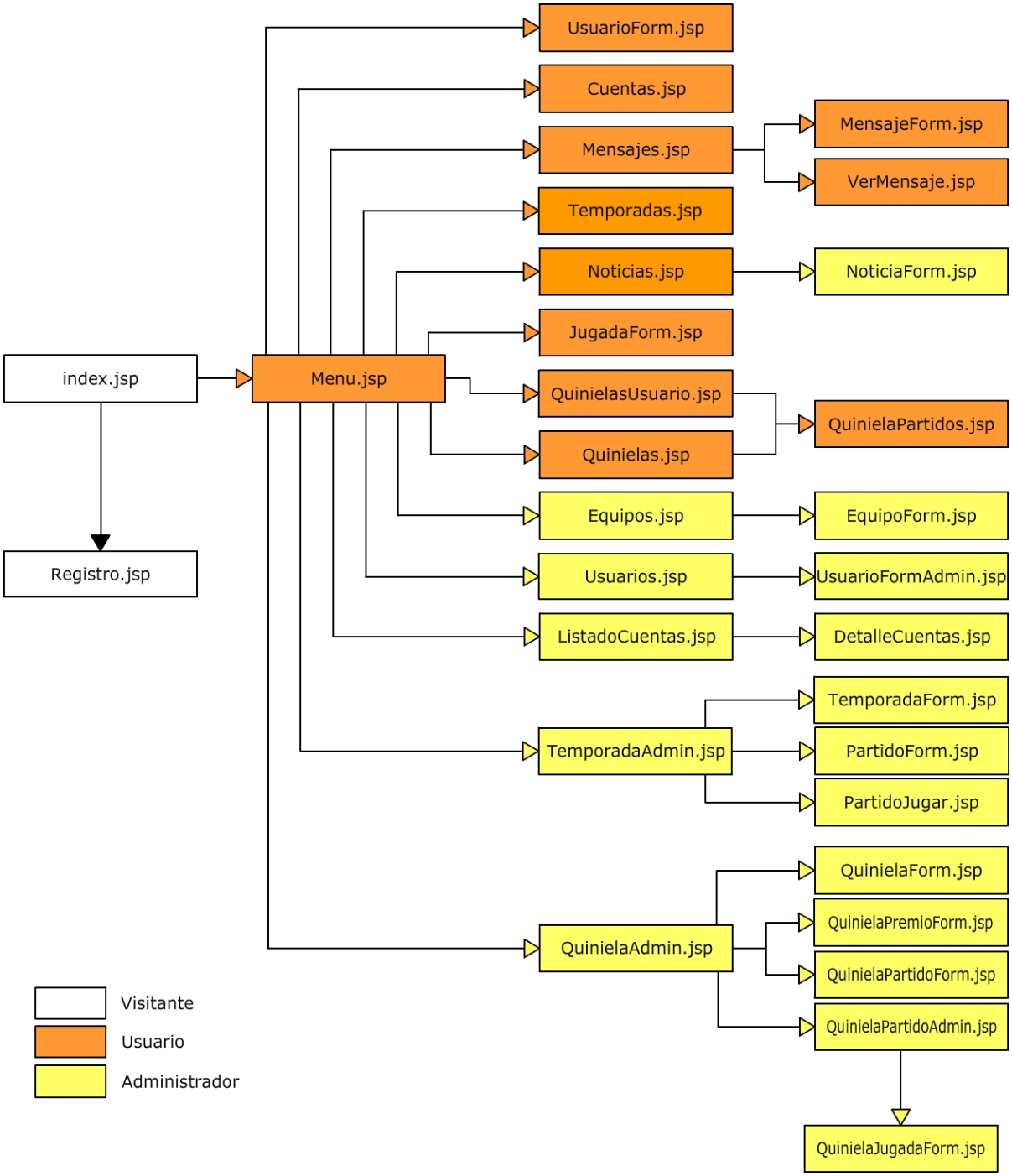
* ENVIAR QUINIELA



* GESTIÓN USUARIOS (ALTA)



3.5.- Mapa Web de la aplicación.



Como podemos ver en el gráfico anterior, las páginas marcadas en amarillo sólo son accesibles para los administradores, mientras que todas las páginas marcadas en naranja son accesibles tanto para los usuarios como para los administradores. Está claro que en nuestro caso, los administradores también participan como un usuario más, enviando sus quinielas y viendo la información que deseen.

3.6.- Instalación del entorno.

Mientras que al iniciar el proyecto, he utilizado diversas herramientas (Ant, XDoclet, JBoss, etc.), el entorno final ha sido instalado en un solo paso utilizando el IDE Eclipse. Concretamente, he utilizado la versión que integra un servidor de aplicaciones JBoss, que incluye tanto la base de datos Hypersonic, como el servidor Tomcat entre otros. Esta versión del IDE Eclipse integra también la herramienta XDoclet para la generación de los documentos XML.

He realizado la descarga del paquete desde la página oficial del JBoss, concretamente en la sección: http://labs.jboss.com/portal/jbossas/download.

La versión utilizada es la **JBoss 4.0.5 GA**. Para hacer funcionar todo el entorno, he hecho algunas modificaciones en el sistema, las cuales se reducen a añadir las variables de entorno necesarias para el servidor de aplicaciones.

El resto de modificaciones han tenido lugar en el Eclipse, en el que he configurado el servidor JBoss como servidor por defecto y he hecho otras configuraciones que tenían como fin el crear los documentos XML y algunas clases java como las destinadas a la interfaz de usuario (EurokiloFacadeLocalHome, EurokiloFacadeLocal, EurokiloFacadeRemoteHome, EurokiloFacadeRemote). La configuración de esta parte del entorno es bastante amplia, por lo que no entraré en detalles.

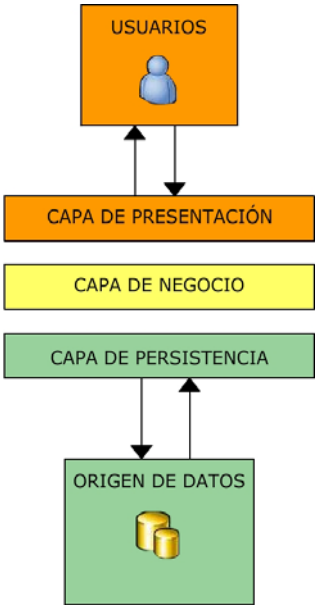
Cabe destacar también la importancia de dos archivos que debemos añadir a la configuración del servidor JBOSS. Se trata de los archivos 1X2-DS.XML (base de datos), 1X2-LOGIN-CONFIG-SERVICE.XML y 1X2-LOGIN-CONFIG.XML (configuración de seguridad). El primero de estos archivos contiene la definición de la base de datos, tanto el usuario y contraseña como la cadena de conexión, el nombre de la base de datos, etc. Los otros dos archivos se refieren a la seguridad JAAS (Servicio de Autenticación y Autorización de Java), y son los encargados de gestionar la entrada de los usuarios mediante el login y el password. Estos tres archivos son esenciales para el correcto funcionamiento de la aplicación. La situación de cada uno de ellos está explicada en la guía de instalación que se adjunta con el código de la aplicación en un archivo comprimido.

Es importante señalar también, que en el código, concretamente en la clase ControllerServlet, existe una constante llamada TEMPORADA_ACTUAL, que contiene el valor del identificador de la temporada en la que nos encontramos. Este valor debe modificarse cuando se inicie una nueva temporada para que los datos mostrados se refieran a esa misma temporada. En sucesivas actualizaciones, se cambiará esta forma de activar una temporada para pasar a una propiedad del objeto TemporadaDTO en la que se establezca si está activa o no. Por falta de tiempo la he tenido que dejar de esta forma temporalmente.

4.- Diseño de la aplicación.

4.1.- Arquitectura.

Para este tipo de aplicación, y aprovechando la tecnología utilizada, he optado por un diseño de triple capa. En este tipo de diseño, lo que se intenta es que cada capa sea totalmente independiente de la otra, con lo que la tarea de actualización se hace muy sencilla. A continuación se detalla de forma resumida cada una de las tres capas, pasando luego a una descripción más interna de las mismas.



- Capa de presentación: Esta capa comprende la interfaz del usuario. Todo aquel elemento susceptible a ser manipulado por el usuario, es presentado en esta capa. Así por ejemplo podemos agrupar las páginas JSP y las hojas de estilo CSS.

- Capa de persistencia: En esta capa se encuentran los datos de la aplicación. Digamos que prácticamente está formada por la base de datos que prepara sus datos para ser controlados por la capa de negocio y para ser enviados por este último a la capa de presentación.

- Capa de negocio: En esta capa se encuentra toda la lógica de negocio. Desde aquí recogemos las peticiones del usuario para gestionarlas y realizar las tareas necesarias como recoger datos de la capa de persistencia y mostrarlos en la capa de presentación o simplemente modificar datos de la capa de persistencia. En nuestro caso, si un usuario quiere ver el resultado de una quiniela concreta, deberá enviar la petición a la capa de negocio, ésta consultará en la capa de persistencia los datos relacionados con esa quiniela y los pasará de nuevo a la capa de presentación que se encargará de formatearla para mostrarla al usuario.

4.1.1.- Capa de presentación.

La capa de presentación sigue un esquema similar al de la aplicación. Se trata de un esquema MVC (Model, View, Controller), en el que volvemos a tener cierta independencia entre cada uno de los elementos. Con este esquema, conseguimos separar el modelo de datos de la aplicación de su representación de cara al usuario. Así, se pueden describir estos elementos de la siguiente forma:

- View: Muestra al usuario la información que necesita. A vista del usuario se trata de simples páginas HTML, mientras que en la parte interna de la aplicación, se trata de páginas JSP (*Java Server Pages*), capaces de recibir los datos del controlador y darles el formato adecuado.

- Model: Contiene la lógica de negocio. Esta capa comprende los objetos POJO (*Plain Old Java Obect*) y los EJB (*Enterprise Java Bean*). En nuestro caso se trata de los EJB.

- Controller: Recibe e interpreta la interacción del usuario, provocando cambios en los datos y en su representación. Esta capa está formada por los Servlet (ControllerServlet en nuestro caso), que están escritos totalmente en Java.

Con esta arquitectura, podemos enumerar una serie de pasos cada vez que el usuario interactúa con la aplicación:

1.- La acción que realiza al usuario llega al servidor mediante un mensaje HTTP.

2.- El mensaje generado en el punto anterior, lo recoge el Servlet, que convierte la petición a lenguaje Java y la envía al Modelo (EJB).

3.- El modelo recibe la petición del Servlet por medio de unas interfaces (EurokiloFacade en nuestro caso) que ocultan su complejidad. Trata la petición y devuelve el resultado al controlador.

4.- El controlador recibe el resultado de su petición por parte del modelo, añade los datos necesarios para su transformación a HTML y devuelve el conjunto a la vista.

5.- La vista recibe el resultado de su primera petición y le da un formato legible para el usuario en formato HTML.

Las ventajas que supone este tipo de esquema es la posibilidad de reutilizar el código del modelo en otras aplicaciones, se trate de aplicaciones Web o no. Otra ventaja es la de separar la lógica del negocio de la vista para el usuario, permitiendo la programación individual de estas últimas sin tener que tocar el código Java de la parte interna de la aplicación.

4.1.2.- Capa de negocio.

La capa de presentación, descrita en el apartado anterior, se comunica directamente con esta otra capa, la capa de negocio. En ésta última es donde residen todas las clases que se encargan de gestionar las peticiones del usuario y acceder a la base de datos para satisfacer sus necesidades.

En esta capa, se sitúan los objetos EJB y el ServiceLocator. Los primeros son los encargados de recoger las peticiones del Servlet desde la capa de presentación, tratar los datos y realizar la conexión con la base de datos mediante el ServiceLocator. Este último necesita de la resolución de nombres JNDI para realizar dicha conexión.

Como ejemplo de esta capa, se puede ver el proceso que lleva a un usuario a ver el resultado de los partidos de una determinada jornada. Para empezar, el usuario realiza la petición [Ver Partidos] al Servlet, que la transforma en código Java y la lleva ante el EJB (EurokiloFacadeLocalHome). Éste recibe la petición y mediante los objetos DAO, que realizan la conexión con la base de datos por medio del ServiceLocator, recupera la información de los partidos pedidos por el usuario. Una vez tiene el resultado, lo devuelve en formato Java al Servlet, que se encarga de mostrarlo al usuario realizando las operaciones necesarias para que sea legible por éste último.

4.1.3.- Capa de persistencia.

La capa de persistencia es quizá la más sensible de la aplicación, ya que dejando a un lado todo el código, es la que contiene los datos que realmente importan en una aplicación Web.

En nuestro caso, las clases que se encargan de gestionar esta capa son los objetos DAO, concretamente los Hibernate-DAO que implementan las interfaces. Estos objetos son tratados como contenedores que recuperan la información de la base de datos física y los almacenan para posteriormente ser tratados. No hace falta decir que la base de datos es una parte importante de esta capa, en la que los objetos reciben la característica de persistentes.

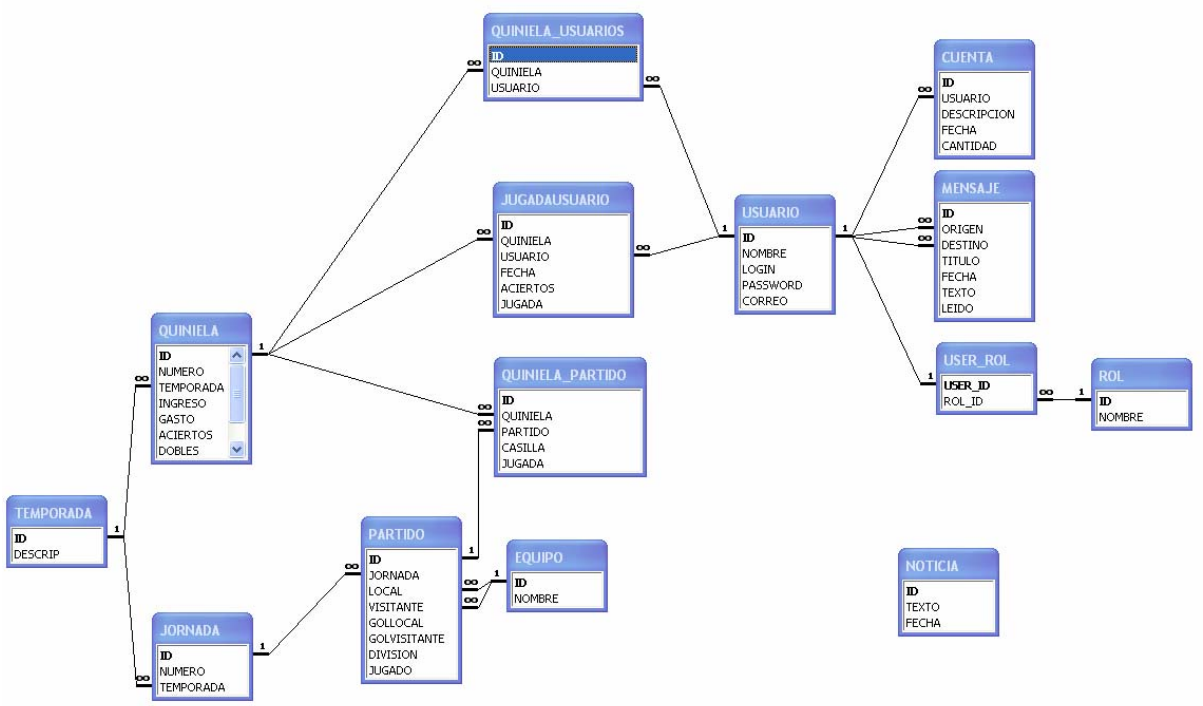
La decisión de implementación para esta capa ha sido la de emplear la base de datos Hypersonic y un API auxiliar llamado Hibernate, el cual me ha ahorrado el uso de sentencias SQL para trabajar con los datos. Mediante este API, se mapean los objetos DTO directamente con la base de datos, por lo que al acceder a cualquier propiedad de estos objetos, lo que se hace realmente es acceder a la base de datos.

Objetos como HibernateUsuarioDAO o HibernateQuinielaDAO entre otros, son los encargados de implementar los métodos de las interfaces (UsuarioDAO o QuinielaDAO) y recoger los datos necesarios mediante la clase ServiceLocator, que crea las conexiones con la base de datos y que, al inicio del proyecto, creaba los

DataSet que devolvían los resultados. Especifico lo del "inicio del proyecto" porque finalmente he optado por el uso de Hibernate para evitar esas consultas SQL almacenadas en un DataSet que accedían de una forma más compleja a la base de datos. El uso de esta API ha despejado muchas líneas de código, ya que al acceder al objeto, accedemos a sus atributos de la base de datos y no son necesarios los DataSet intermedios. En el próximo apartado se detalla la estructura de la base de datos, con lo que se pueden ver las propiedades de los objetos DTO.

4.3.- Diseño de la base de datos (persistencia)

Para almacenar los datos, como dije anteriormente, he utilizado Hypersonic como base de datos. Esta BD viene integrada en el sistema JBOSS, de ahí mi decisión de utilizarla. Las tablas creadas se detallan a continuación:



Los nombres de las tablas son los suficientemente descriptivos para no necesitar explicación. Por otro lado, se pueden observar dos relaciones importantes.

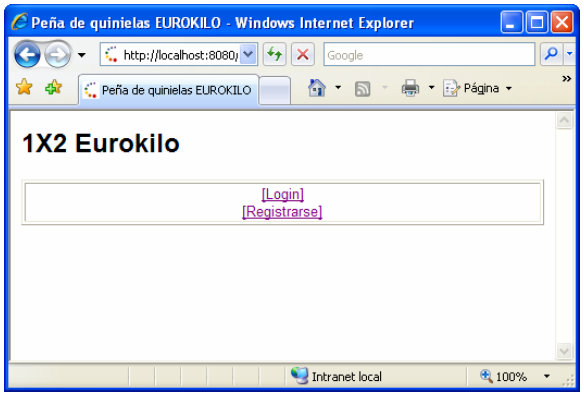
Por un lado, la tabla QUINIELA_USUARIOS, contiene los usuarios que han participado en una quiniela determinada.

Y por otro lado, la tabla QUINIELA_PARTIDO, contiene los partidos que forman una quiniela. Esta relación tiene un atributo añadido, que es la JUGADA que se realizó en la quiniela correspondiente.

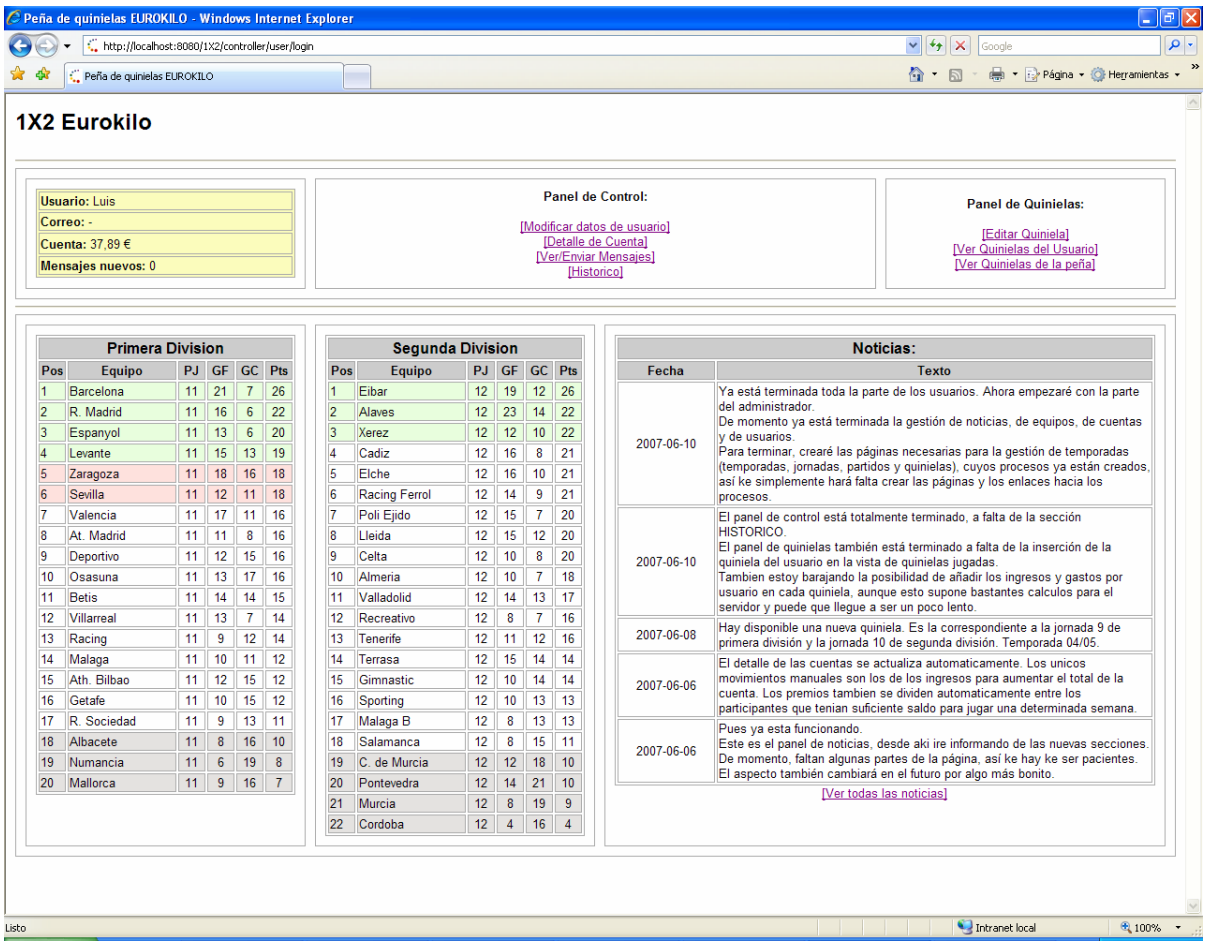
4.4.- Diseño de la interfaz de usuario (presentación)

El diseño de las páginas no ha sido muy cuidado, ya que le he dado más valor a los datos y a su presentación que al formato de las páginas. Igualmente, en las siguientes capturas muestro las imágenes de las páginas más relevantes. Su situación se puede observar en el mapa Web expuesto anteriormente.

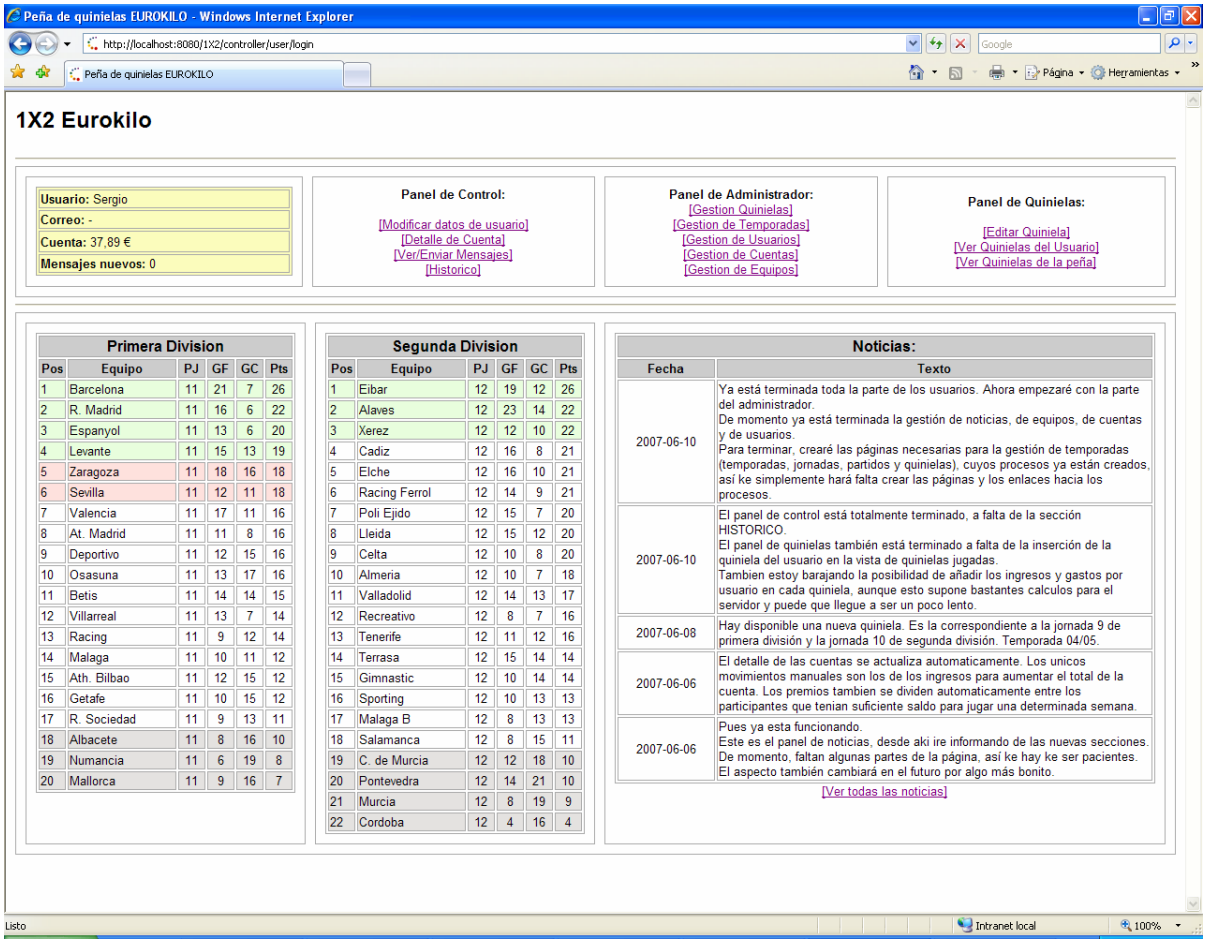
Index.JSP



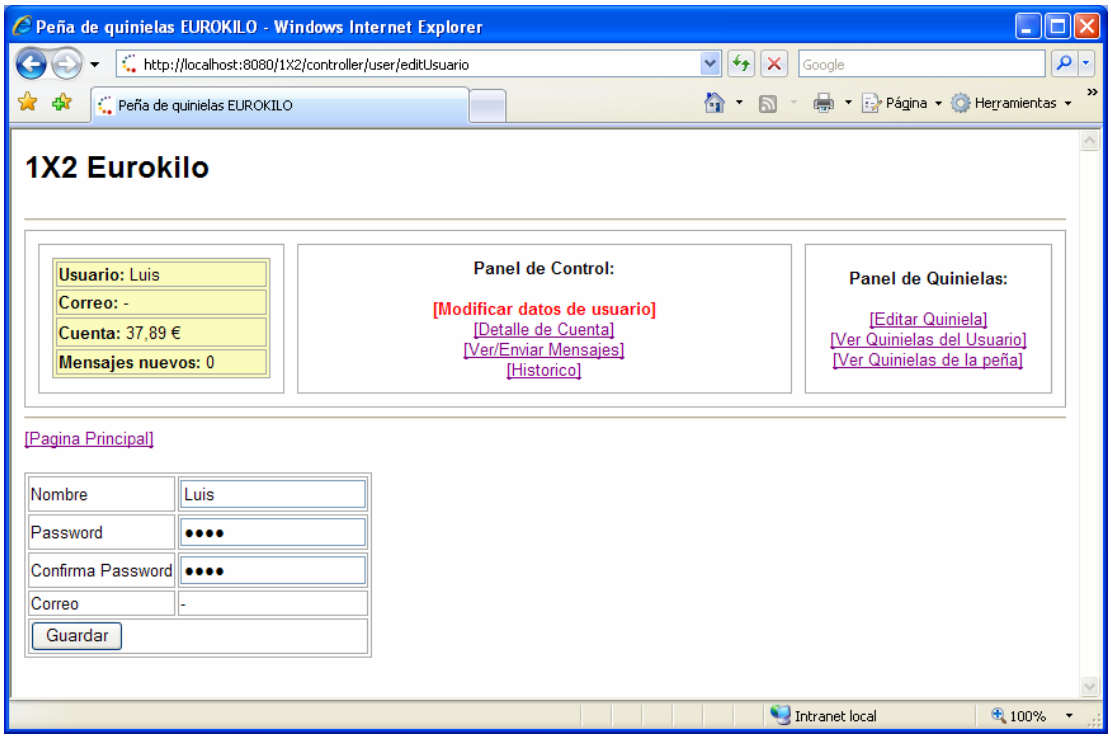
Menu.JSP (versión usuario)



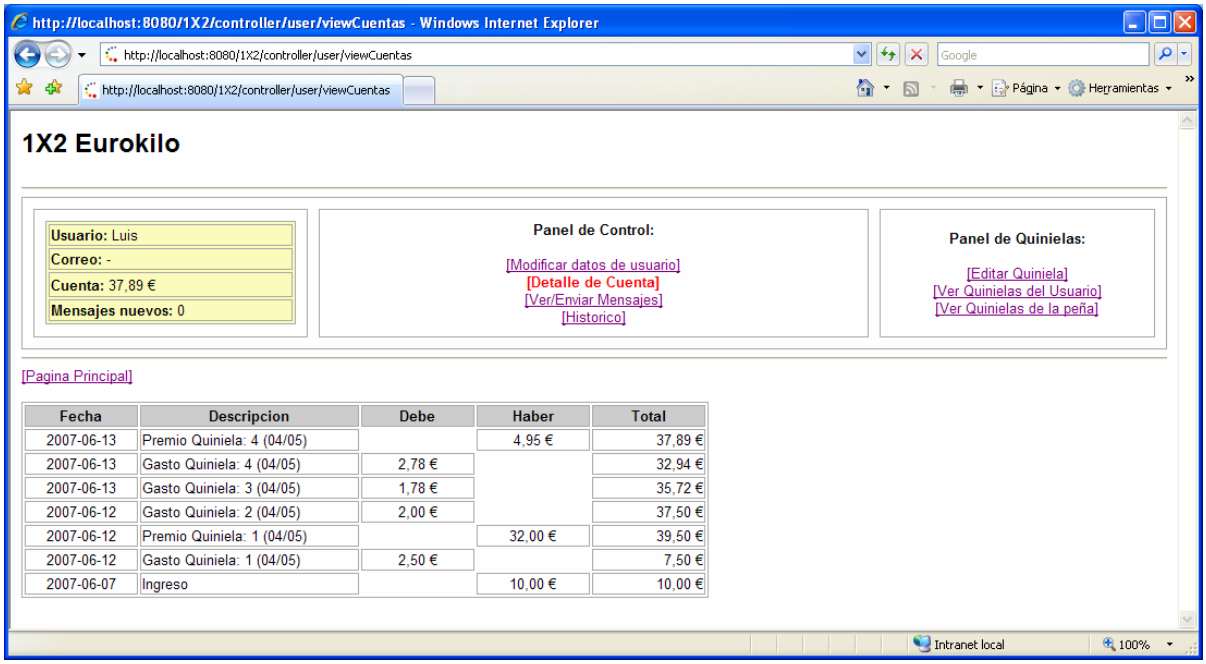
Menu.JSP (versión administrador)



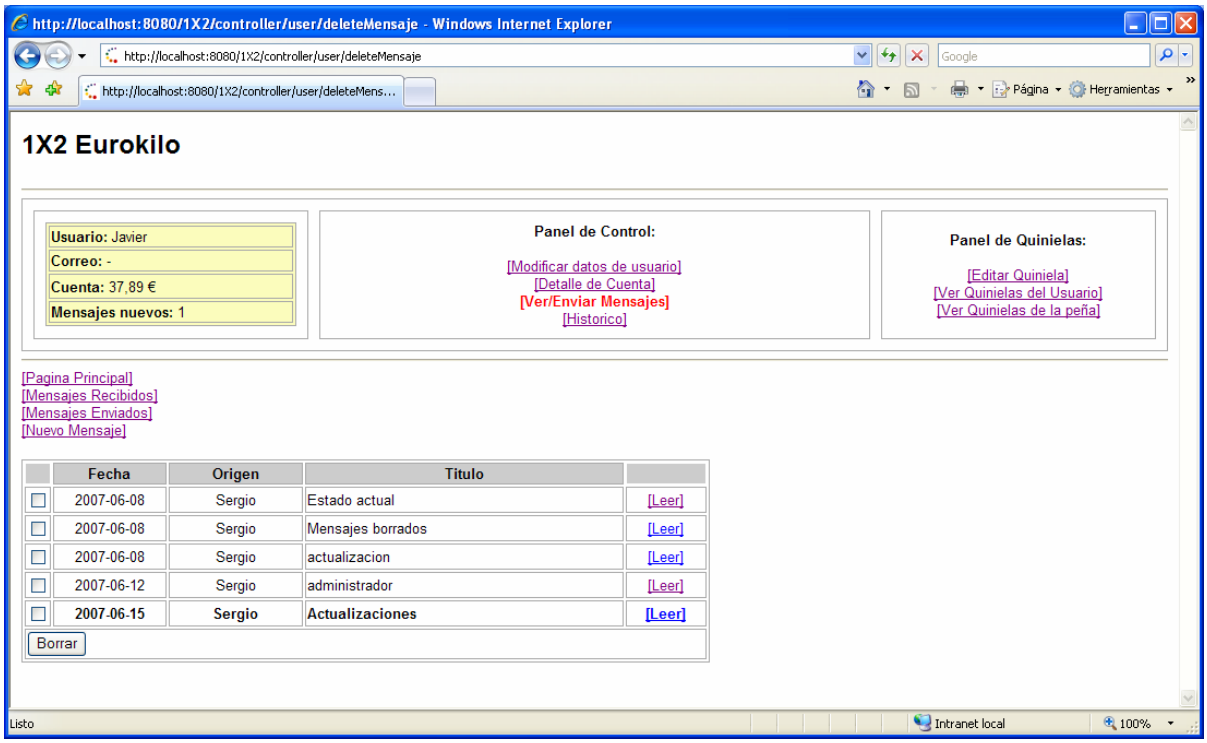
UsuarioForm.JSP



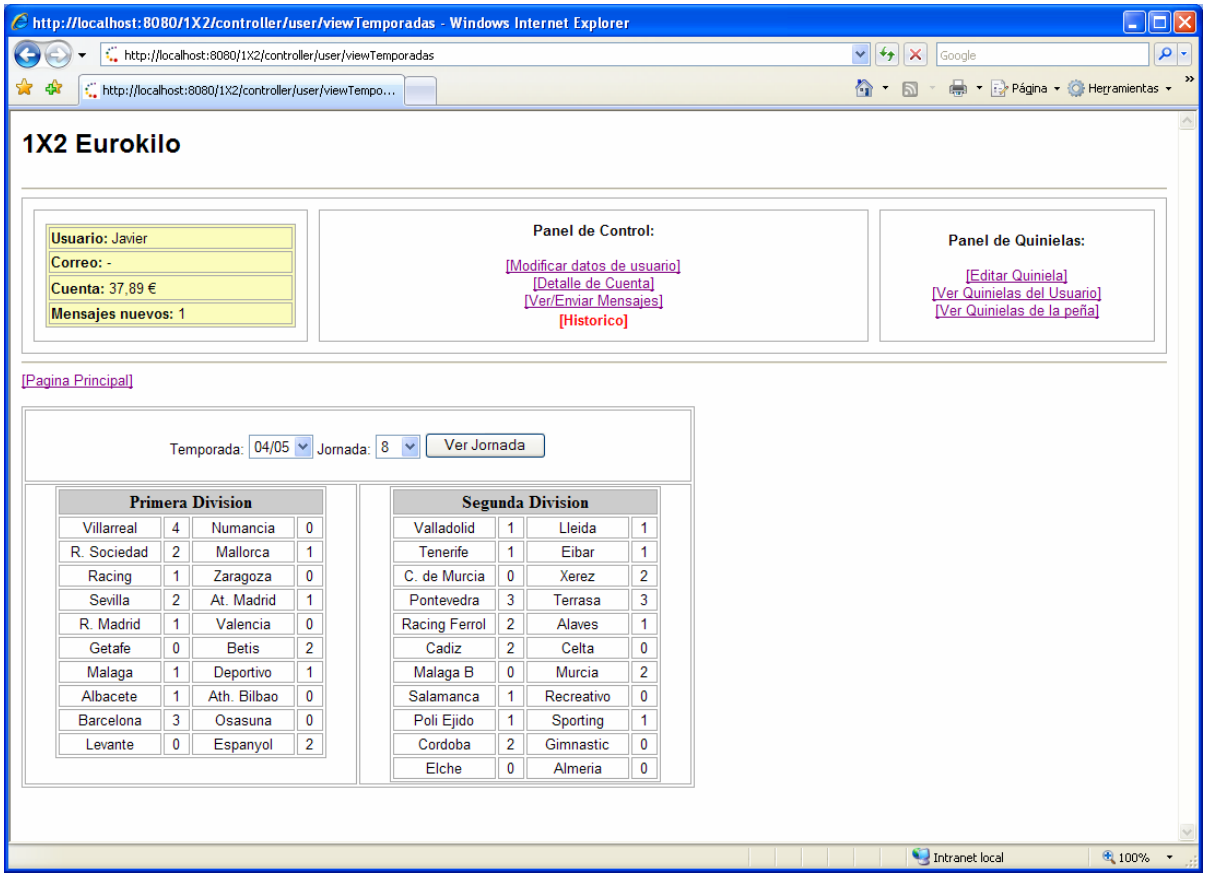
Cuentas.JSP



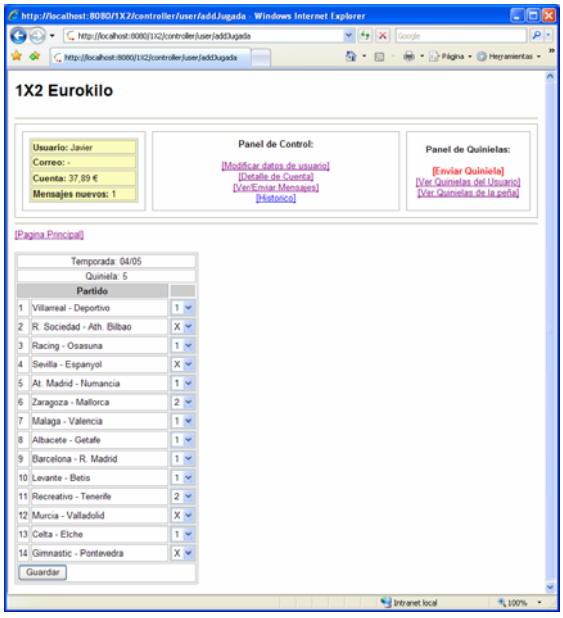
Mensajes.JSP

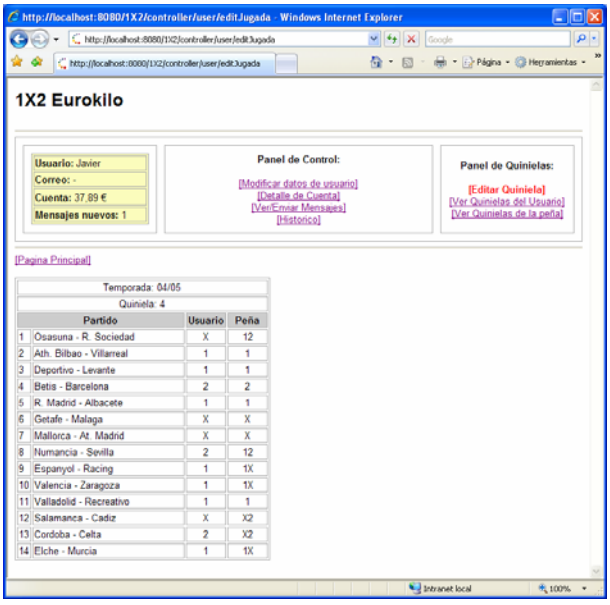


Temporadas.JSP

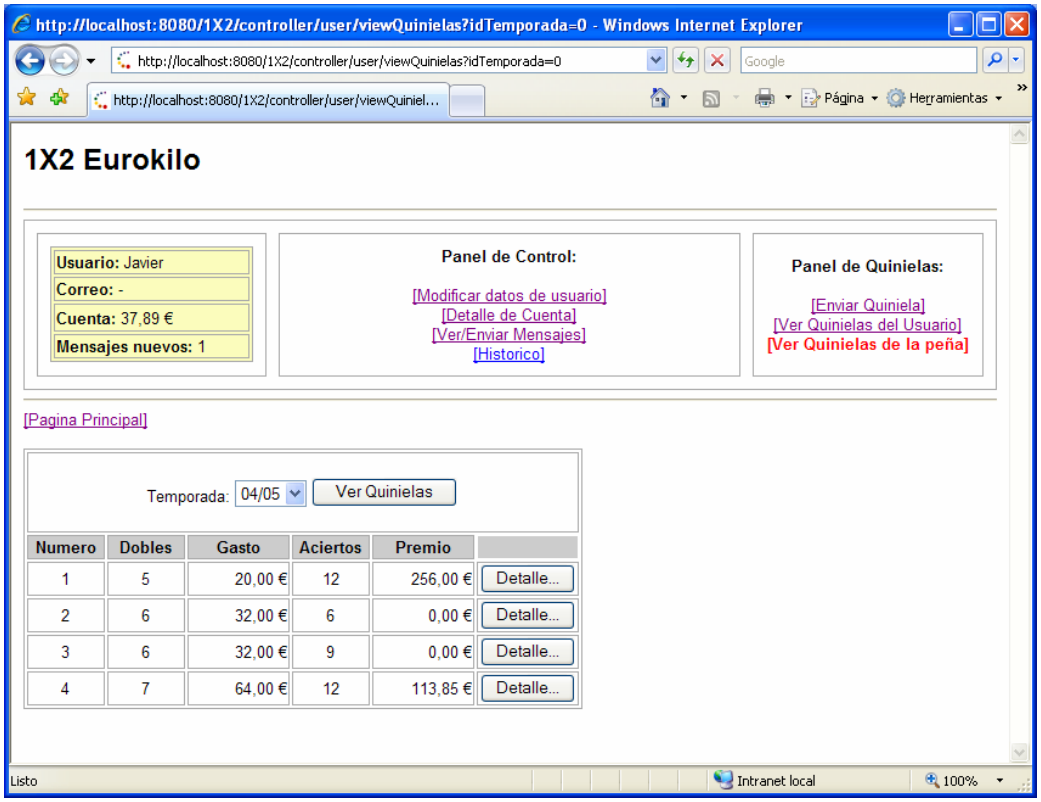


JugadaForm.JSP (Con quiniela abierta y con quiniela cerrada)

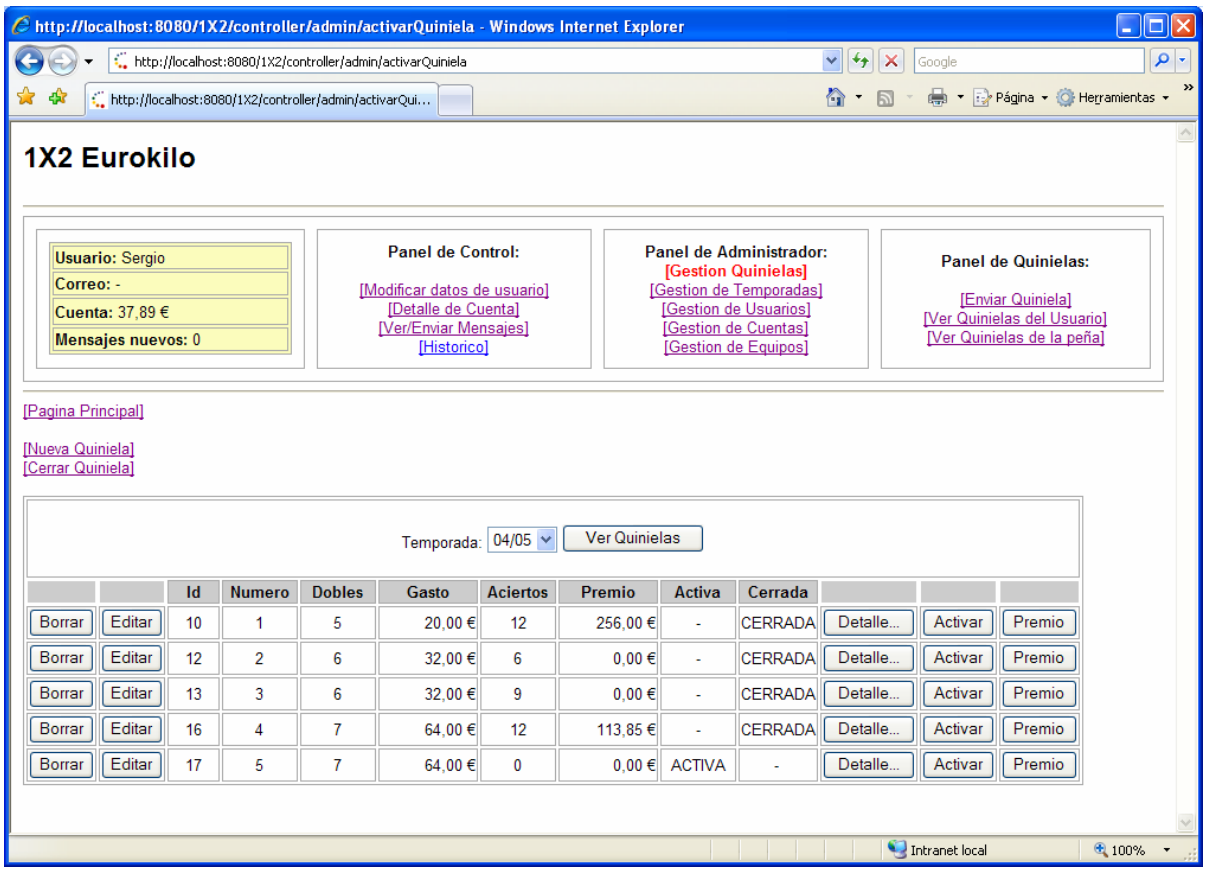




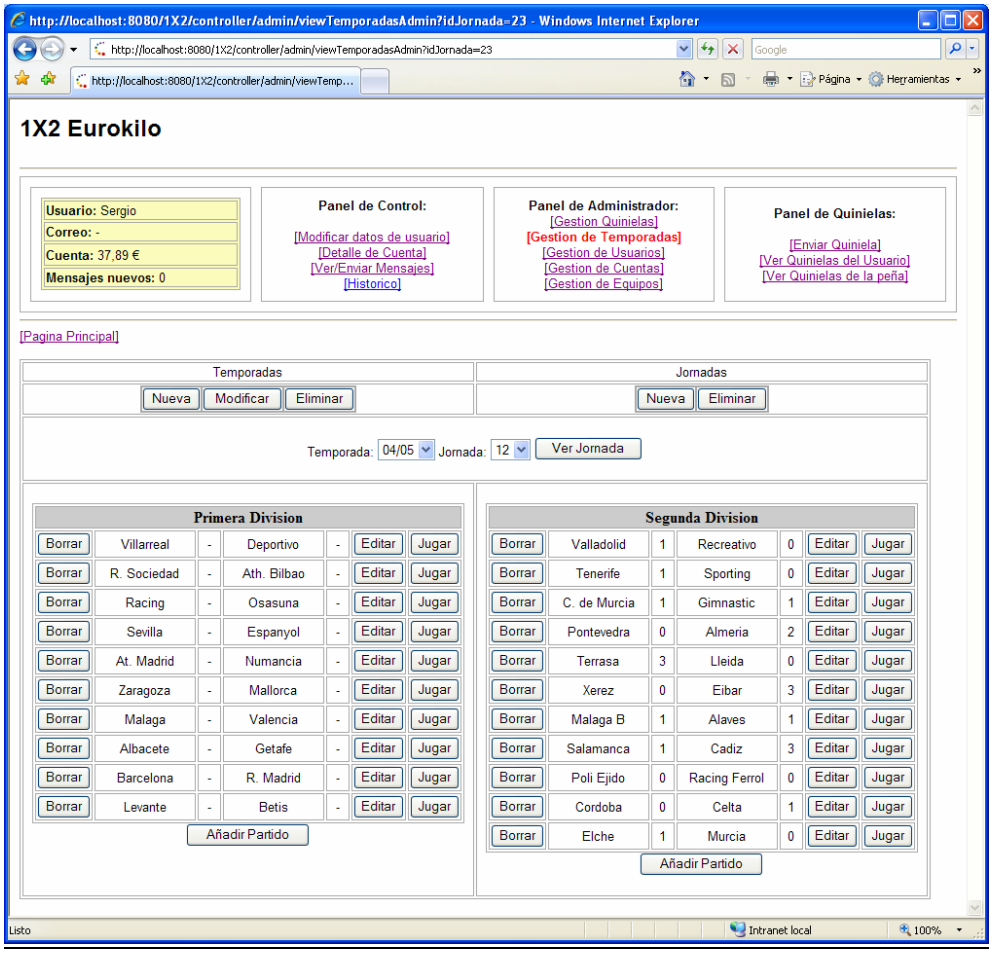
Quinielas.JSP



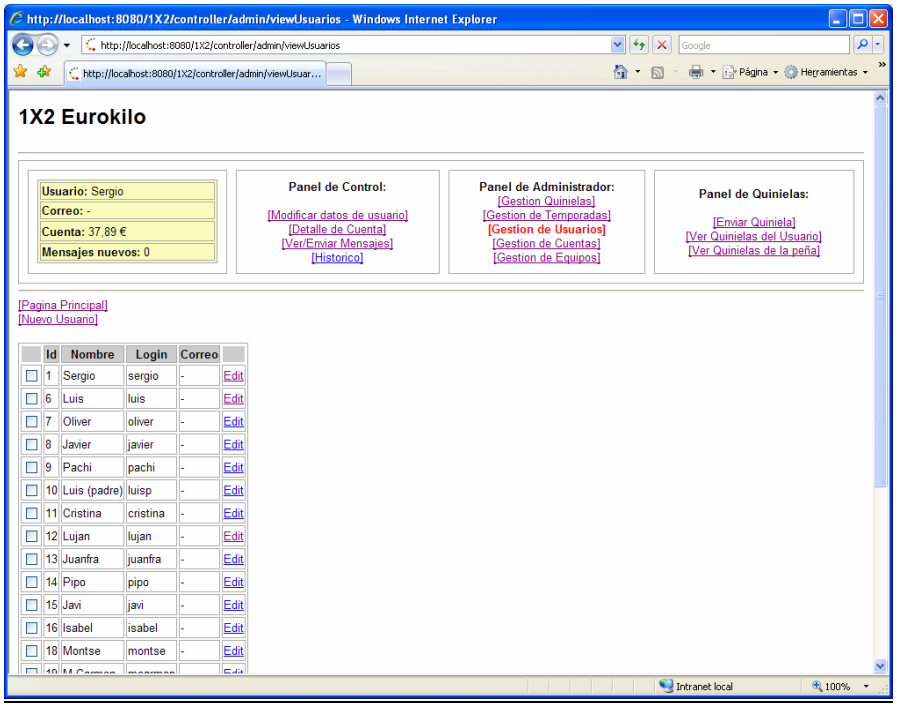
QuinielaAdmin.JSP



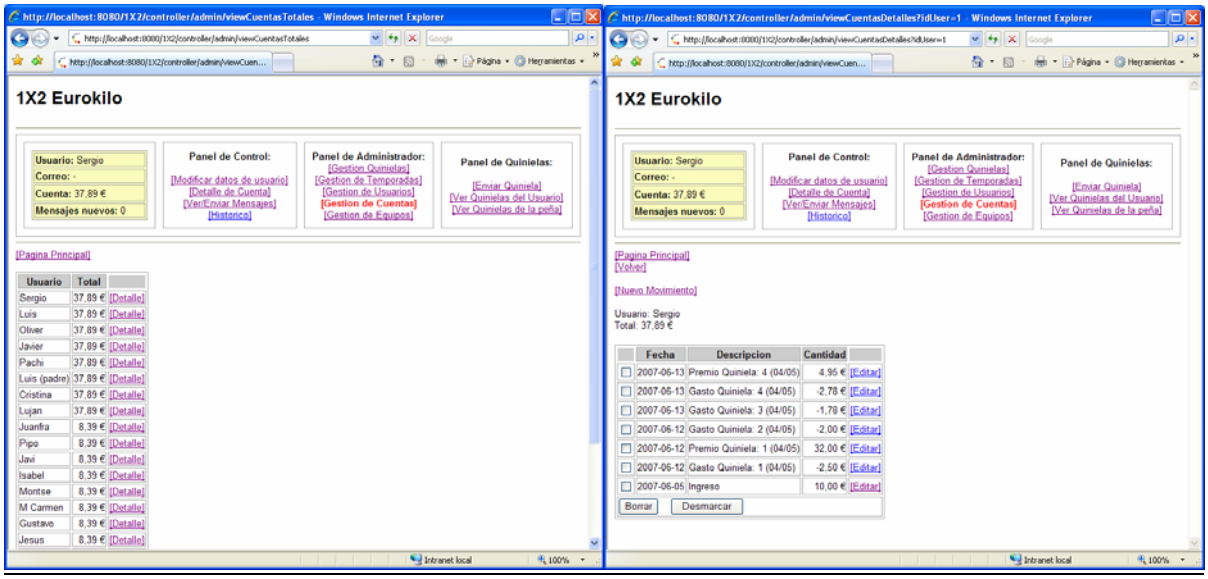
TemporadaAdmin.JSP



Usuarios.JSP



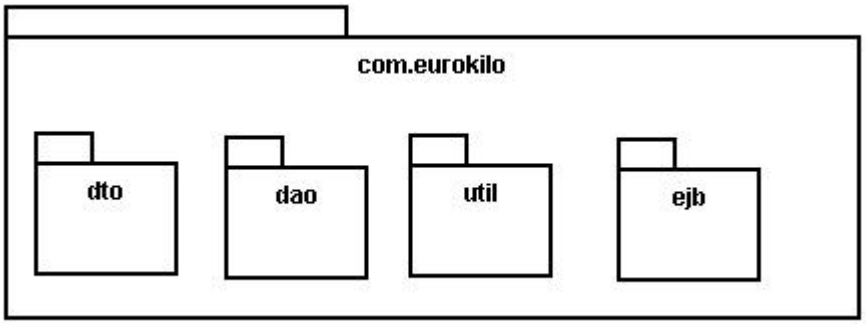
ListadoCuentas.JSP // DetalleCuentas.JSP



4.5.- Diagrama de clases (negocio)

Por último, detallo las clases java utilizadas en el proyecto describiendo la jerarquía de clases en el formato UML.

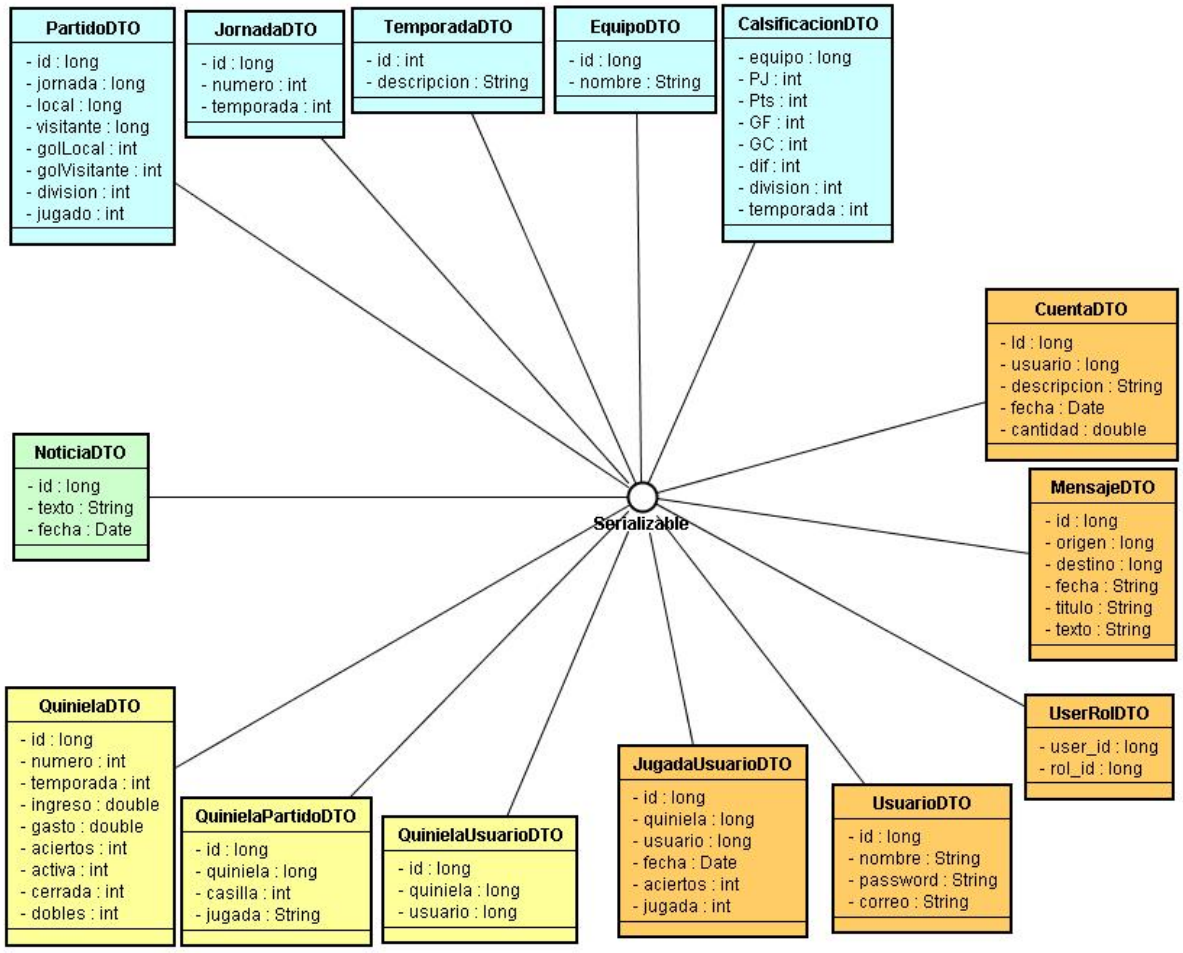
Para empezar se muestra la estructura de paquetes, para luego pasar a detallar cada uno de ellos.



Package com.eurokilo

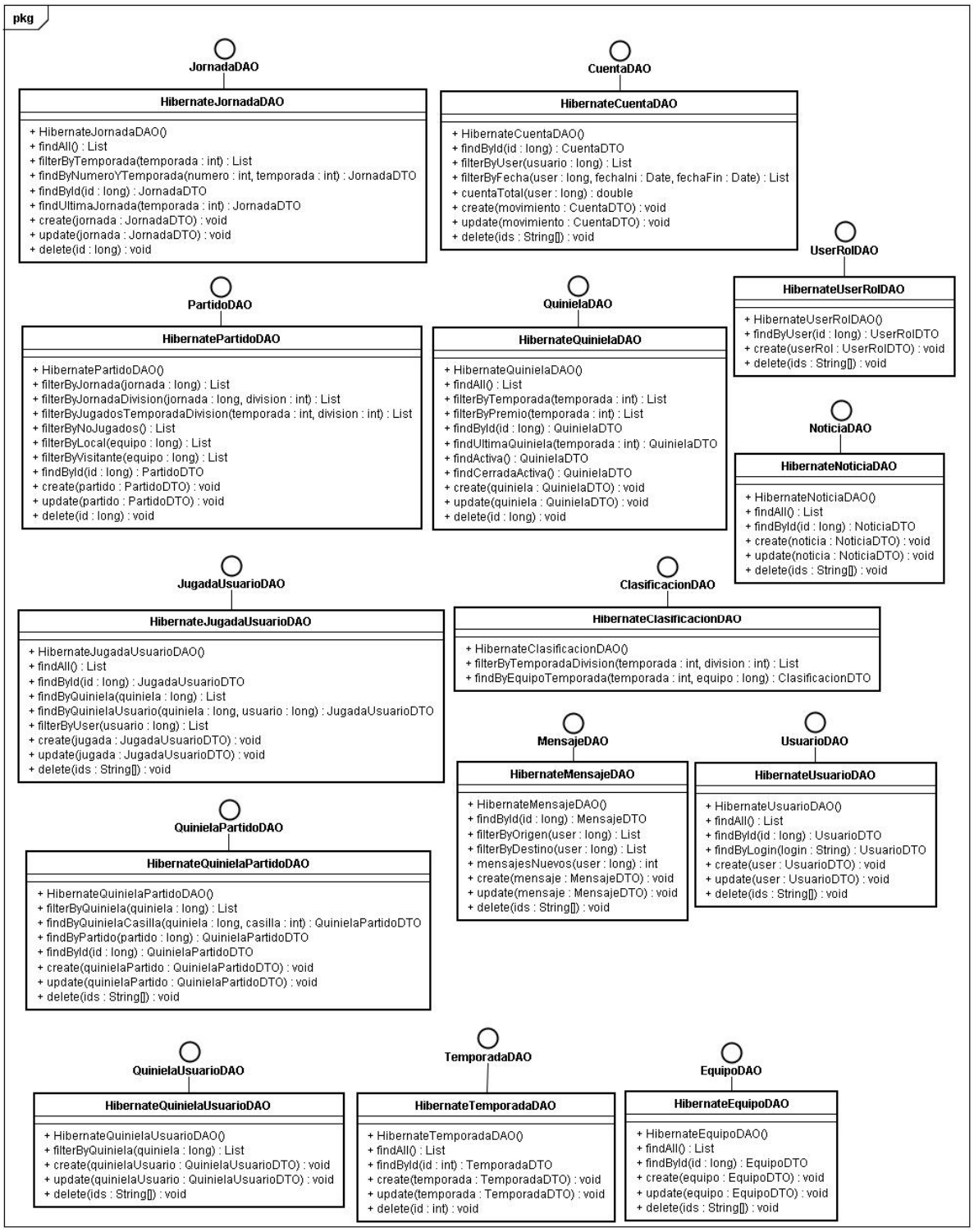
Package com.eurokilo.dto

Todas las clases tienen sus correspondientes getters y setters.

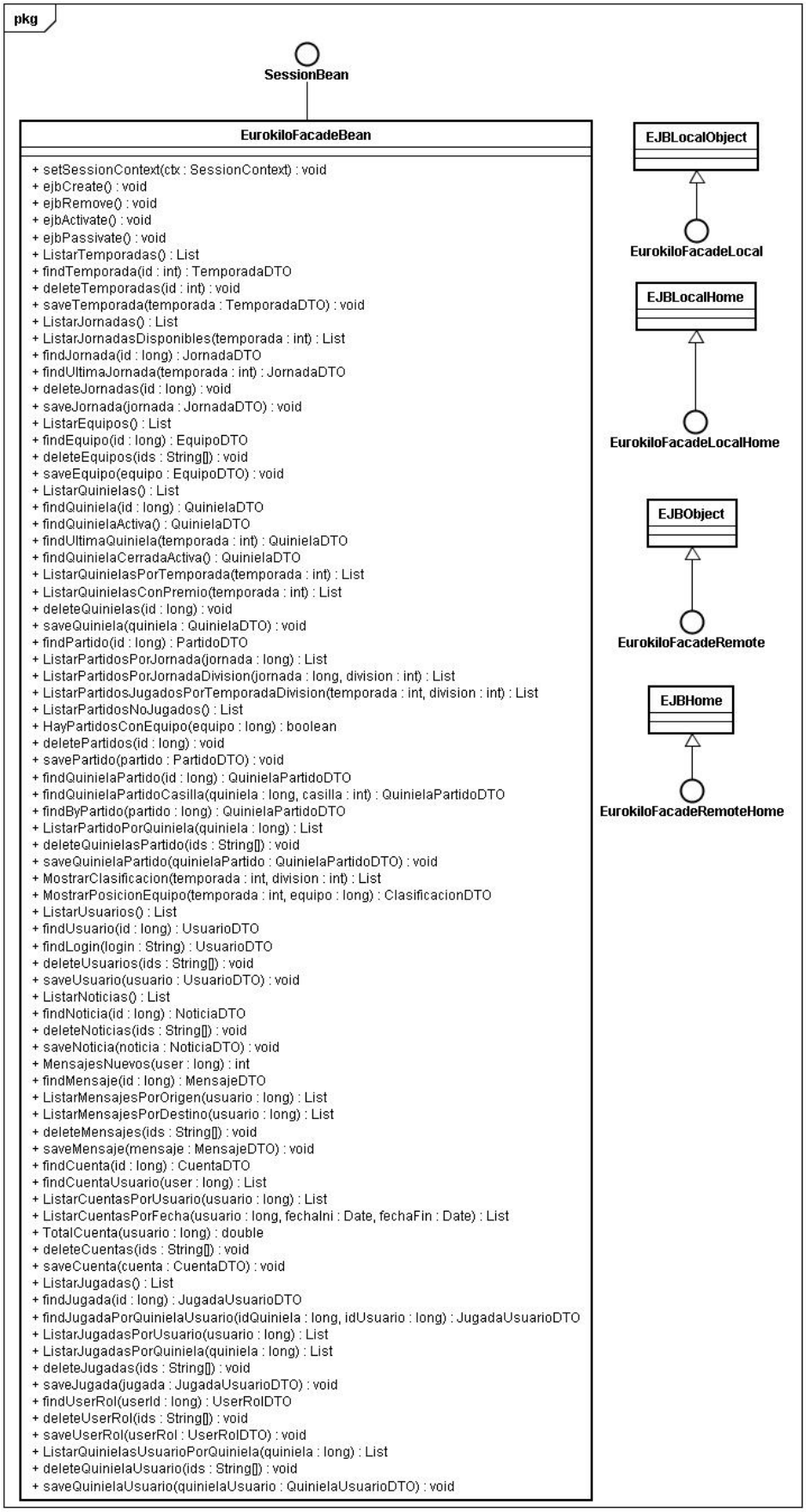


Package com.eurokilo.dao

Todas las clases del paquete implementan la constante:
HIBERNATE_SESSION_FACTORY : String = "java:comp/env/hibernate/SessionFactory



Package com.eurokilo.ejb



Package com.eurokilo.util