**PROYECTO FINAL DE CARRERA**

# Sistema de gestión de ligas

**Autor:** Pau Lozano Lloret
**Director:** Félix Buendía García
**Curso:** 2009-2010

# Contenido


**1 – INTRODUCCIÓN 4**

**1.1 – OBJETIVOS 4**
**1.2 – CONTEXTO 5**
**1.3 – ESTRUCTURA DEL DOCUMENTO 7**

**2 – ESPECIFICACIÓN DE REQUISITOS 8**

**2.1 – INTRODUCCIÓN 8**
2.1.1 – PROPÓSITO 8
2.1.2 – ÁMBITO 8
2.1.3 – DEFINICIONES, ACRÓNIMOS Y ABREVIATURAS 9
2.1.4 – REFERENCIAS 10
**2.2 – DESCRIPCIÓN GENERAL 11**
2.2.1 – PERSPECTIVA DEL PRODUCTO 11
2.2.2 – FUNCIONES DEL PRODUCTO 11
2.2.3 – CARACTERÍSTICAS DEL USUARIO 12
2.2.4 – RESTRICCIONES GENERALES 13
**2.3 – REQUISITOS ESPECÍFICOS 14**
2.3.1 – REQUISITOS DE INTERFACES EXTERNAS 14
2.3.1.1 – INTERFACES DE USUARIO 14
2.3.1.2 – INTERFACES HARDWARE 14
2.3.1.3 – INTERFACES SOFTWARE 14
2.3.2 – REQUISITOS FUNCIONALES 15
2.3.2.1 – INTERFAZ WEB 15
2.3.2.2 – INTERFAZ J2ME 18
2.3.3 – REQUISITOS DE EFICIENCIA 19
2.3.4 – RESTRICCIONES DE DISEÑO 19
2.3.5 – ATRIBUTOS 19

**3 – ANÁLISIS 20**

**3.1 – INTRODUCCIÓN 20**
**3.2 – CASOS DE USO 21**
**3.3 – DIAGRAMA DE CLASES 22**
**3.4 – ANÁLISIS DE GESPARTIDOS 23**
3.4.1 – CASOS DE USO 23
3.4.2 – DIAGRAMA DE CLASES 24

**4 – DISEÑO 25**

**4.1 – ARQUITECTURA DEL SISTEMA 25**
**4.2 – CAPA DE PRESENTACIÓN 27**
4.2.1 – DISEÑANDO LA INTERFAZ 27

4.2.2 – INTERFAZ DE LA PARTE PÚBLICA 28
4.2.3 – INTERFAZ DE LA PARTE PRIVADA 32
**4.3 – CAPA DE LÓGICA DE NEGOCIO 37**
**4.4 – CAPA DE PERSISTENCIA DE DATOS 41**
4.4.1 – DIAGRAMA ENTIDAD-RELACIÓN 41
**4.5 – DISEÑO DE GESPARTIDOS 44**
4.5.1 – DISEÑO DE LA INTERFAZ 44
4.5.2 – LÓGICA DE LA APLICACIÓN 47

**5 – IMPLEMENTACIÓN 49**

**5.1 – TECNOLOGÍAS 49**
5.1.1 – TECNOLOGÍAS EMPLEADAS EN GESPARTIDOS 51
**5.2 – HERRAMIENTAS 53**
**5.3 – DETALLES DE IMPLEMENTACIÓN 54**
5.3.1 – IMPLEMENTACIÓN DE GESPARTIDOS 63

**7 – PRUEBAS 68**

**7.1 – PRUEBAS DE GESPARTIDOS 70**

**8 – CONCLUSIONES 71**

**9 – BIBLIOGRAFÍA 72**

**10 – ANEXOS 74**

**10.1 – SENTENCIAS SQL PARA LA CREACIÓN DE LAS TABLAS DE LA BD 74**
**10.2 – CÓMO PROTEGER EL DIRECTORIO DE ADMINISTRACIÓN EN APACHE 76**
**10.3 – CÓMO FIRMAR UN *MIDLET* 80**

# 1 – Introducción

Este documento describe el trabajo realizado para el proyecto final de carrera que consiste en crear un sistema de gestión global de ligas que llamaremos GesLigas. Servirá para cualquier deporte que se juegue por parejas, como por ejemplo frontenis, tenis, pádel, pilota valenciana.

Se accederá a la aplicación a través de una interfaz web y los datos se guardarán en una base de datos, con todo esto podremos instalarla en un servidor y acceder a ella desde cualquier parte a través de Internet. Además incorporará un asistente implementado sobre J2ME (Java 2 Micro Edition) para gestionar los partidos en la misma pista de juego mediante un dispositivo móvil que soporte dicha tecnología. A este asistente lo llamaremos GesPartidos.

## 1.1 – Objetivos

El propósito general de la aplicación es gestionar de forma integral un tipo de liga concreto en el que se compite en partidos por parejas pero se computan los tantos de forma individual.

Además se pretende que la cronología de los partidos sea lo más flexible posible. Tan flexible que no habrá una programación previa: se generará la lista de los próximos partidos a jugar en la misma pista de juego en función de los jugadores que asistan a la jornada. El asistente GesPartidos instalado en un dispositivo móvil generará esta la lista de partidos teniendo en cuenta el registro de partidos que ya se han jugado e incluso lo hará de forma dinámica, es decir, que se podrá regenerar la lista en mitad de la jornada en caso de que varíe el conjunto de jugadores asistentes. En la misma lista se podrán anotar los resultados de cada partido y se guardarán en un fichero que se podrá subir al servidor desde la interfaz web para actualizar la base de datos.

A través del portal web, los participantes y aficionados podrán consultar los resultados, ranking, estadísticas y la normativa. Pero además habrá un calendario donde se anunciarán los eventos relacionados con el torneo y un pequeño chat-foro que servirá como punto de encuentro virtual donde los usuarios podrán expresar cualquier opinión o comentario sobre la liga.

## 1.2 – Contexto

Hay muchos gestores de ligas y torneos en general de cualquier deporte, sobre todo de fútbol.

Los hay de tipo genérico que sirven para muchos deportes como el de Sagois SLU al que llaman "Competiciones Deportivas". Es bastante completo e incluso se encarga también de gestionar la parte económica de la competición. La gestión se realiza en un ordenador en el que se instala la aplicación y es posible también generar páginas de resultados en HTML. Puede descargarse una versión de demostración desde su web *www.sagois.com/compdep/* .

Otro genérico y gratuito es Splendid City (en *www.splendidcity.net* ). Está desarrollado en Java, gestiona múltiples torneos a la vez y de muchos tipos. Ofrece cierta automatización en la gestión del calendario y exporta también a HTML y XML, entre otros. Integra además un gestor de contactos, un cliente de e-mail y un cliente de FTP que podemos usar para subir los documentos que se generan.

Y otro similar a los anteriores es AutoLiga, del que se puede descargar una versión de evaluación desde www.autoliga.net .

Algo más parecido al proyecto GesLigas son los gestores online como www.tenisbase.com , www.esportalia.com , www.micompeticion.com y www.deportesreunidos.com . Son portales web que te permiten crear y gestionar tu propio torneo online de forma gratuita, tan solo hay que registrarse. Estas herramientas son algo más sencillas pero mucho más accesibles no solo para los administradores de la liga sino también para cualquier jugador o aficionado que desee informarse sobre el evento. La última de ellas (Deportes reunidos), además aporta un enfoque diferente: se autodefine como una "red social deportiva". Se integra con la mayoría de las redes sociales existentes y, además de organizar y gestionar torneos, permite encontrar y conocer gente con la que competir jugando.


*Figura 1: Captura de pantalla de www.esportalia.com*


*Figura 2: Captura de pantalla de www.tenisbase.com*


*Figura 3: Captura de pantalla de www.micompeticion.com*

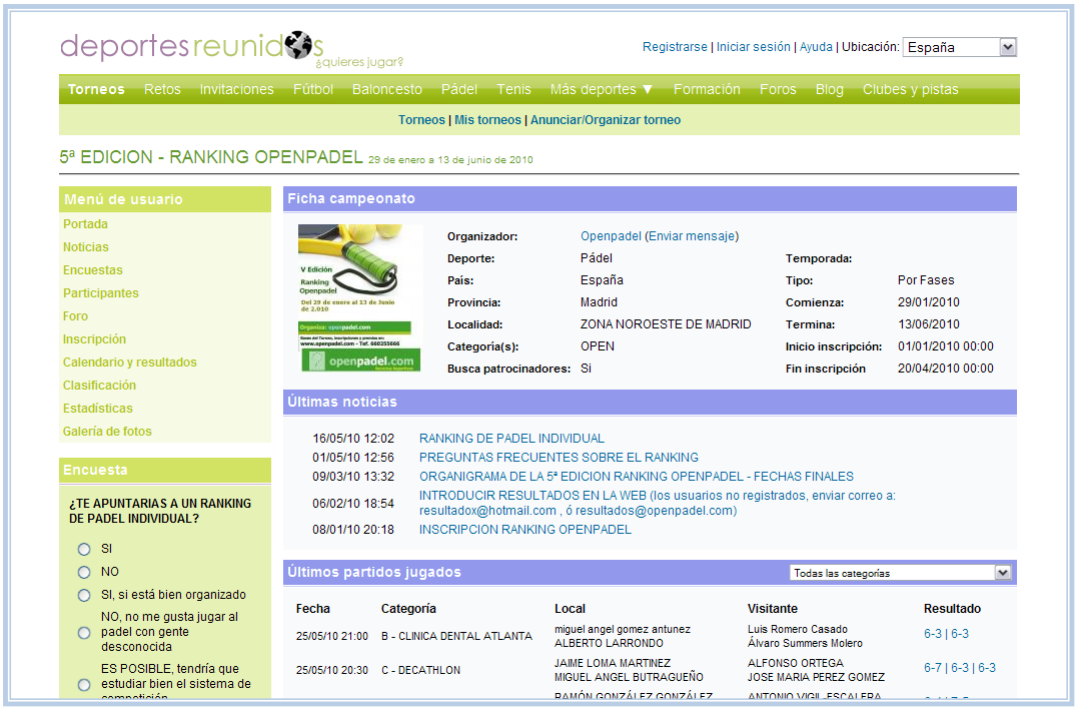


*Figura 4: Captura de pantalla de www.deportesreunidos.com*

## 1.3 – Estructura del documento

El presente documento es la memoria del trabajo realizado para el proyecto y está organizado en distintos capítulos que se describen a continuación.

El desarrollo de cualquier aplicación informática se debe realizar en cinco fases: especificación de requisitos, análisis, diseño, implementación, integración y pruebas. Los siguientes capítulos corresponden a cada una de estas fases.

En el capítulo 2 se utiliza un estándar IEEE para redactar la especificación de requisitos software de la aplicación a desarrollar. Dicho estándar ayuda a extraer los requisitos de un producto definiendo la estructura que debe tener el documento donde quedan plasmados.

En el capítulo 3, a partir de un análisis de los requisitos, se determina qué elementos intervienen en el sistema, así como su estructura, relaciones y como se va a comportar. Todo esto queda reflejado en diagramas del estándar UML. En esta fase del desarrollo ya se separa, en el último apartado, el análisis específico para el asistente GesPartidos.

Una vez se tiene claro qué va a realizar la aplicación, en el capítulo 4 se determina cómo lo va a hacer. Se implementará en tres partes o capas: presentación, lógica y persistencia. En el apartado de presentación se mostrará el diseño de la interfaz de usuario mediante capturas de pantalla. En el apartado de lógica de negocio se explicará el comportamiento interno de la aplicación según el tipo de página en la que nos encontremos detallando la interacción y el trasiego de datos con las otras dos capas. En el siguiente apartado se explica cómo se organiza la capa de persistencia y cómo se estructuran los datos mediante un diagrama entidad-relación.

En el último apartado se presenta el diseño de la herramienta de gestión de los partidos en pista. En este caso la arquitectura es más simple y no se desarrolla en capas separadas. En el primer subapartado se muestra la apariencia que tendrá más o menos la herramienta y en un segundo subapartado se muestra su diseño interno de forma estática y dinámica.

En el capítulo 5 (*Implementación*) se explica qué tecnologías y herramientas se han empleado en el desarrollo del proyecto y se muestran los detalles más relevantes de la implementación de GesLigas en sus distintas capas y también del asistente GesPartidos.

En el capítulo 6 se describen las pruebas que han sido realizadas para verificar el buen funcionamiento de la aplicación.

Por último se expondrán algunas conclusiones y propuestas para futuras ampliaciones.

Al final se incluye una referencia del material y la documentación consultada que ha servido para el desarrollo del proyecto y también algunos anexos.

# 2 – Especificación de requisitos

En este capítulo se expone la especificación de requisitos software del proyecto que está basada en el estándar IEEE 830-1998, un formato comúnmente usado en el ámbito de los proyectos software.

## 2.1 – Introducción

### 2.1.1 – Propósito

Con el presente documento se pretenden formalizar todas y cada una de las funcionalidades que debe tener la aplicación de gestión de ligas junto con los usuarios que la van a usar. Este documento representa el canal de comunicación entre las distintas partes implicadas, define por completo la aplicación deseada y constituye así la base sobre la cual el desarrollador puede planificar el proyecto y diseñar la aplicación.

### 2.1.2 – Ámbito

El producto a desarrollar se puede clasificar como software de gestión. Se trata de un sistema de gestión global de ligas que debe funcionar para cualquier deporte en el cual se jueguen partidas en las que intervengan dos equipos de dos jugadores cada uno.

El funcionamiento de las ligas que se gestionarán es del tipo “todos con todos contra todos”, es decir, se jugarán todos los partidos que se puedan formar con el conjunto de los jugadores. Cada jugador formará pareja con cada uno del resto de jugadores para competir contra todas las parejas que se puedan formar con los demás jugadores (todos menos los dos primeros). Cada pareja jugará solo una vez con cada una del resto de parejas, dicho en otras palabras, cada posible partido se jugará solo una vez.

Para 4 jugadores (A, B, C y D) el número de partidas en la liga sería de 3.

AB-CD; AC-BD; AD-BC

Para 5 jugadores (A, B, C, D y E) el número de partidas sería de 15.

AB-CD; AB-DE; AB-CE; AC-BD; AC-BE; AC-DE; AD-BC; AD-BE; AD-CE; AE-BC; AE-BD; AE-CD; BC-DE; BD-CE; BE-CD;

Y así para cualquier número de jugadores.

Con cada partido ganado se suman 2 puntos al jugador. Y al final, cuando se han disputado todos los partidos, el jugador que tenga más puntos gana la liga.

Este sistema tiene la ventaja de ser muy flexible en la programación de los partidos, pues no impone restricciones de orden. Por otra parte, se pretende “evaluar” a cada jugador

individualmente aunque se juegue por parejas. Pero tiene un inconveniente: el número de partidos a jugar crece mucho con cada jugador que se añade, de tal forma que a partir de 8 o 9 jugadores se hace demasiado largo el torneo. Está pensado para pocos jugadores (entre 4 y 7) con interés en jugar durante una temporada larga.

El sistema constará de un portal web accesible desde cualquier navegador por Internet y una aplicación para dispositivos móviles que soporten J2ME (Java 2 Micro Edition).

El portal web mostrará toda la información relevante del estado de la liga, jugadores, equipos y partidos jugados. Habrá un calendario de eventos programados en la liga, como por ejemplo partidos o sesiones de entrenamiento, y se expondrá también la normativa con la que se rige la liga. Por último habrá un pequeño chat-foro que servirá como punto de encuentro para tratar temas relacionados con la liga donde cualquier persona podrá escribir sus comentarios. El coordinador de la liga podrá realizar todas las gestiones y la edición de contenidos también a través del portal web pero mediante un menú de administración solo accesible por personal autorizado.

La aplicación auxiliar Java se instalará en un dispositivo móvil y permitirá al coordinador de la liga gestionar los partidos en la misma pista de juego. Desde la web se podrá descargar la base de datos en un fichero en formato XML. De esta forma la aplicación Java tendrá acceso a la base de datos y creará la lista de partidos a jugar en la jornada partiendo del histórico de partidos ya jugados para que no se repitan y en el orden adecuado para que todos los jugadores tengan oportunidad de jugar por igual. La generación de los partidos será totalmente dinámica, ya que se hará en función de los jugadores presentes teniendo en cuenta que podrán llegar e irse jugadores a lo largo de la jornada. Por lo tanto no habrá una programación estricta de partidos sino que cada jugador podrá asistir cuando pueda a las jornadas de liga. Por último, la aplicación permitirá además apuntar los resultados para generar, al final, un fichero directamente en formato de sentencias SQL que podrá cargarse cómodamente desde la web (accediendo al menú de administración en cualquier ordenador conectado a Internet) para actualizar la base de datos. En el momento en que actualicemos la base de datos en Internet, la web ya mostrará los datos del ranking y demás estadísticas actualizados de acuerdo a los últimos resultados.

Aunque en la gestión de partidos, el coordinador estará asistido por la aplicación Java en su terminal móvil, siempre se podrán realizar las gestiones oportunas (introducir resultados de partidos, introducir/eliminar jugadores, etc.) manualmente desde el menú de administración de la web que se ha mencionado antes.

### 2.1.3 – Definiciones, acrónimos y abreviaturas

**IEEE**: Institute of Electrical & Electronics Engineers.

**FIV**: Facultad Informática Valencia.

**ERS**: Especificación de Requerimientos Software.

**J2ME**: Java 2 Micro Edition. Es una versión reducida para dispositivos móviles de la plataforma Java de Sun Microsystems. Permite desarrollar aplicaciones portables en un lenguaje de alto nivel como es Java.

**XML**: eXtensible Markup Language (lenguaje de marcas extensible). Es un metalenguaje extensible de etiquetas desarrollado por el W3C que permite, en este caso, especificar el formato del fichero de intercambio de la base de datos.

**SQL**: Structured Query Language (lenguaje de consultas estructurado). Es un lenguaje que permite expresar operaciones de consulta y modificación en una base de datos relacional.

**XHTML**: eXtensible HyperText Markup Language (lenguaje extensible de marcas de hipertexto). Es un lenguaje de marcado para la construcción de páginas web. Permite definir tanto la estructura como el contenido en texto, imágenes y demás componentes de las páginas web.

### 2.1.4 – Referencias

Para la elaboración de este documento se han consultado los siguientes documentos:

- Ejemplos de IEEE 830.
- IEEE Std. 830-1998. Guía del IEEE para la ERS.
- Germán Vidal y Javier Jaén. Práctica 1. Ingeniería de Requerimientos. Curso 06-07. FIV.

## 2.2 – Descripción general

### 2.2.1 – Perspectiva del producto

El producto a desarrollar se trata de un portal web que, en definitiva, son un conjunto de páginas web que genera un servidor web dinámicamente de acuerdo a los datos que obtiene del servidor de bases de datos. Conforme le son solicitadas, el servidor web las envía a cada ordenador de cada usuario donde el explorador web se encargará de mostrarlas en pantalla.

GesLigas no será parte o componente de otro software, será un producto en sí mismo, sin embargo dependerá del servidor web, el servidor de bases de datos y del explorador con que se visualice. La ventaja es que no depende del sistema operativo y si se usan bien los estándares que existen las dependencias se resuelven más fácilmente.

### 2.2.2 – Funciones del producto

Las funciones que debe realizar el producto a través de la interfaz web las podemos clasificar en varios bloques:

a) Funciones de visualización de estado
   - Mostrar ranking
   - Mostrar calendario
   - Mostrar tabla de equipos
   - Mostrar recuento de partidas

b) Funciones de introducción de datos (solo para administradores)
   - Introducir jugador
   - Introducir partido
   - Introducir evento en calendario
   - Modificar evento en calendario
   - Eliminar evento en calendario
   - Introducir separador en calendario

c) Funciones de importación/exportación de datos (solo para administradores)
   - Guardar base de datos en fichero XML
   - Introducir jornada a partir de fichero SQL

d) Función adicional de desactivar/reactivar jugador (solo para administradores)
   - Desactivar jugador
   - Reactivar jugador

e) Función de visualización de normativa
   - Mostrar normativa y funcionamiento de la liga

f) Funciones de foro
   - Mostrar foro
   - Enviar post

Las funciones que debe realizar el producto a través de la interfaz en J2ME (que solo usarán los administradores) se enumeran a continuación por orden cronológico (pasos del asistente):

1) Cargar base de datos desde fichero XML
2) Seleccionar jugadores presentes
3) Generar partidos
4) Anotar resultado de partido
5) Cambiar jugadores presentes y regenerar partidos (y volvemos al paso 4)
6) Guardar jornada en fichero SQL

### 2.2.3 – Características del usuario

En esta sección vamos a ver qué tipo de usuarios van a usar el producto y como afectan estos a las funciones que debe realizar el producto.

Hay varios tipos de usuarios que usarán las distintas partes del sistema y cada uno de ellos requiere distintas funcionalidades. En general todos los usuarios poseen conocimientos básicos de navegación web.

En primer lugar están los jugadores que participan en la liga. Estos requieren toda la información sobre el estado general de la liga: tanto su posición particular (sus puntos, el resultado de sus partidas) como también la información referente a sus oponentes. Además querrán conocer estadísticas e información derivada que les ayude a analizar y planificar su trayectoria de juego. Tienen que estar al tanto de cada uno de los eventos que se producen en el trascurso de la liga a través del calendario. Podrán consultar la normativa. Y, por último, tendrán un pequeño punto de encuentro virtual en el chat-foro para comentar cualquier cosa en relación con la liga.

Los aficionados y cualquier persona que acceda a la web podrán acceder a toda la información, como los jugadores, y escribir mensajes en el chat-foro.

Los administradores de la liga, además de todo lo mencionado anteriormente, requieren un menú de administración desde el que poder realizar todas aquellas operaciones sobre la base de datos, necesarias para la correcta gestión del torneo. Desde este menú tendrán que introducir cada jugador, introducir los partidos que se van disputando y gestionar los eventos del calendario. En la pista de juego tendrán que decidir qué partidos se juegan en cada momento procurando la máxima alternancia entre los jugadores presentes para no favorecer ni perjudicar a ninguno de ellos. Y además, tendrán que anotar los resultados de cada partido para luego introducirlos en la base de datos. Todo esto lo podrán hacer cómodamente con el asistente de GesPartidos.

### 2.2.4 – Restricciones generales

Se debe poder hacer una copia de seguridad del contenido de la base de datos en cualquier momento desde el menú de administración.

En cuanto a la seguridad, se requiere que solo los administradores de la liga puedan acceder al menú de administración de la web por medio de una contraseña y por tanto solo ellos podrán cambiar los datos de la liga en la base de datos.

## 2.3 – Requisitos específicos

### 2.3.1 – Requisitos de interfaces externas

#### 2.3.1.1 – Interfaces de usuario

La interfaz será sencilla. Debe contener al menos un título, un menú y una zona de contenidos. Desde el menú se accederá a cada una de las secciones, que se mostrarán en la zona de contenidos. El menú debe contener al menos las siguientes secciones:

- ranking
- calendario
- normativa
- chat-foro

Cada una de estas secciones corresponde a alguna de las funciones requeridas y por tanto sus contenidos se describen en el apartado *2.3.2 – Requisitos funcionales*.

#### 2.3.1.2 – Interfaces hardware

Para que funcione la aplicación es imprescindible un ordenador conectado en red (normalmente a Internet) sobre el que habrá instalado un servidor web y otro ordenador conectado a este (o el mismo) sobre el que funcionará un servidor de bases de datos. Estos dos servidores serán los que harán todo el trabajo. Normalmente este servicio se contrata y debe ser capaz de soportar toda la demanda de los usuarios, que en principio será poca pues se trata de pequeñas ligas, con pocos seguidores.

Para hacer uso de esta aplicación no se requiere ningún hardware específico por parte de los usuarios (excepto administradores), solamente es necesario un ordenador conectado en red con acceso al servidor web (normalmente estará en Internet).

Los administradores además, para poder usar el asistente de GesPartidos, necesitarán un terminal móvil que soporte aplicaciones J2ME, con MIDP 2.0 y CLDC 1.1. Desde hace algunos años casi todos los teléfonos móviles del mercado soportan esta tecnología.

#### 2.3.1.3 – Interfaces software

GesLigas requiere un servidor web y un servidor de bases de datos. El primero debe ser capaz de interpretar y ejecutar el código de la aplicación para generar las páginas web (ya que será una aplicación web dinámica). Además debe poder realizar operaciones sobre la base de datos y recuperar los resultados. El servidor de bases de datos debe ser capaz de interpretar y ejecutar transacciones en lenguaje SQL.

Para acceder a la web del gestor de ligas se requiere de un explorador web gráfico. Hay exploradores en cualquier sistema operativo y la web se debe poder visualizar correctamente

al menos en los que más se emplean actualmente: Microsoft Internet Explorer 7, Mozilla Firefox 3.6 y Google Chrome 4.1.

### 2.3.2 – Requisitos funcionales

En este apartado se describen detalladamente cada una de las funcionalidades que debe poseer el producto. Se agrupan en bloques y se ordenan del mismo modo que en el apartado anterior *2.2.2 – Funciones del producto*.

#### 2.3.2.1 – Interfaz web

##### 2.3.2.1. a – Funciones de visualización de estado

- **Mostrar ranking**
  Se muestra una tabla con todos los jugadores en orden descendiente con los puntos de cada jugador (primero el que más puntos tiene). Para cada jugador se debe mostrar: nombre, puntos que tiene actualmente, partidos que ha ganado, partidos que ha perdido, partidos que ha jugado y si está activo o desactivado.

- **Mostrar calendario**
  Se muestra una lista de eventos relacionados con el torneo. Para cada evento se muestra la fecha y el mensaje. La fecha es una cadena de texto corta, el administrador ya decidirá el formato de la fecha (por ejemplo puede ser un intervalo de tiempo en lugar de una fecha concreta). El mensaje es una cadena de texto larga.

  En el texto de la fecha y el del mensaje pueden aparecer etiquetas XHTML y estas deben ser interpretadas y representadas en pantalla como corresponde.

  Los eventos puede estar remarcados o pueden ser eventos pasados (se debe indicar con alguna marca).

  Se deben poder añadir separadores entre eventos para indicar distintas épocas o fases del torneo.

- **Mostrar tabla de equipos**
  Se mostrará una tabla con todos los jugadores en las filas y todos los jugadores en las columnas, relacionando así cada jugador con el resto de jugadores. En cada relación se indicará el número de veces que han jugado juntos como equipo y el número de veces que han sido rivales en partidos que ya se han jugado.

- **Mostrar recuento de partidas**
  Esta función muestra el número de jugadores, el número de partidos que se han jugado hasta el momento, el total de partidos a jugar en la liga (varia con el número de jugadores) y el total de partidos por jugador.

#### 2.3.2.1. b – Funciones de introducción de datos

Estas funcionalidades serán accesibles solo para administradores.

- **Introducir jugador**
  Esta función crea un nuevo jugador en la base de datos. Requiere que se introduzca en un campo de texto el nombre del nuevo jugador.

- **Introducir partido**
  Esta función se empleará para introducir el resultado de un partido que se ha jugado. Se debe indicar obligatoriamente la fecha en que se ha jugado, cada uno de los cuatro jugadores que han intervenido y el número de tantos de cada equipo al final del partido.

- **Introducir evento en calendario**
  Con esta función creamos nuevos eventos en el calendario. Se requiere indicar, para cada evento, una cadena de texto corta para la fecha, una cadena de texto larga para el mensaje y si el evento estará remarcado o será un evento pasado.

  En el texto de la fecha y el del mensaje pueden aparecer etiquetas XHTML.

- **Modificar evento en calendario**
  Cada evento del calendario podrá ser modificado. Esta función mostrará en campos editables cada uno de los atributos del evento a modificar (fecha, mensaje y tipo) de la misma forma que al introducir el evento. Al terminar la edición, el evento debe ser modificado en la base de datos.

- **Eliminar evento en calendario**
  Con esta función podremos eliminar cualquier evento del calendario.

- **Introducir separador en calendario**
  Con esta función introducimos un separador en el calendario. Nos puede servir para indicar que se entra en una nueva fase del torneo, por ejemplo, que termina la fase de partidos amistosos y empieza la liga.

#### 2.3.2.1. c – Funciones de importación/exportación de datos

Estas funcionalidades serán accesibles solo para administradores.

- **Guardar base de datos en fichero XML**
  Esta función servirá para hacer copias de seguridad de la base de datos y para transferir la base de datos al dispositivo móvil para que el asistente GesPartidos tenga la información actualizada. Cuando se ejecute esta función, el servidor web generará un fichero en formato XML que podremos descargar en nuestro ordenador.

- **Introducir jornada a partir de fichero SQL**
  Con esta función actualizaremos la base de datos del servidor añadiendo los nuevos partidos que se han jugado y que el asistente GesPartidos ha guardado en un fichero en formato de sentencias SQL. Cuando se ejecute esta función se mostrará una ventana de exploración de directorios en el sistema de ficheros de nuestro ordenador mediante la cual seleccionaremos el fichero con extensión .sql que queramos cargar, i este se enviará al servidor. El servidor Web ejecutará cada sentencia en la base de datos indicando si hubo éxito o falló la actualización.

#### 2.3.2.1. d – Función adicional de desactivar/reactivar jugador

Estas funcionalidades serán accesibles solo para administradores.

- **Desactivar jugador**
  El administrador podrá desactivar jugadores, por ejemplo, si no asisten a las jornadas de liga durante un tiempo prolongado o si se retiran del torneo. Cuando se desactiva un jugador se marcará convenientemente allá donde aparezca en la web y se anularán todos aquellos partidos en los que haya intervenido; esto significa que estos partidos ya no computarán a efectos de puntuación para ningún jugador y no se incluirán en ningún recuento. Esto debe ser así debido a que el jugador en cuestión no ha jugado todos sus partidos, y en principio no lo va a hacer, lo que podría favorecer o perjudicar a los jugadores con los que ya ha jugado.

- **Reactivar jugador**
  Tal como se desactivan los jugadores, podrán reactivarse en cualquier momento recuperando los partidos que ya hayan jugado y que volverán a incluirse en el recuento de partidos y puntos para todos los jugadores.

#### 2.3.2.1. e – Función de visualización de normativa

- **Mostrar normativa y funcionamiento de la liga**
  Esta función muestra en pantalla la normativa por la que se rige liga. Esto incluye el reglamento de juego (reglas que deben seguir los jugadores en la pista) y la forma de proceder en el desarrollo de la liga (funcionamiento de la liga, formación de los equipos, adjudicación de puntos, etc.).

  La normativa será una cadena de bastante larga que podrá contener etiquetas XHTML y estas deben ser interpretadas y representadas en pantalla como corresponde.

#### 2.3.2.1. f – Funciones de foro

- **Mostrar foro**
  Al seleccionar la entrada de foro en el menú se mostrará en pantalla un pequeño foro: un listado de mensajes ordenados por la fecha en la que se enviaron. En cada mensaje se mostrará el nombre del autor, el texto del mensaje y la fecha de envío.

  El texto del mensaje podrá contener emoticonos (pequeñas imágenes animadas que representan caras y que ayudan a expresar emociones en el mensaje).

  La lista de mensajes se dividirá en páginas cuando sea muy larga y el usuario debe poder acceder a todas las páginas para seguir el historial de mensajes.

- **Enviar post**
  Cualquier usuario podrá enviar un mensaje desde la misma pantalla en que se muestra el foro, indicando su nombre en un campo de texto corto y su mensaje en un campo de texto largo (pero limitado). Podrá añadir emoticonos en el mensaje. Cuando termine de escribir su mensaje lo podrá enviar al servidor pulsando un botón. Entonces se actualizará la lista de mensajes en pantalla incluyendo el nuevo mensaje.

#### 2.3.2.2 – Interfaz J2ME

Las funciones que debe realizar el producto a través de la interfaz en J2ME (que solo usarán los administradores) se enumeran a continuación por orden cronológico (pasos del asistente):

1) **Cargar base de datos desde fichero XML**
   Para empezar, siempre que se inicia la aplicación se leerá la base de datos desde un fichero contenido en el sistema de ficheros del dispositivo móvil. El fichero estará en formato XML y contendrá todos los jugadores inscritos en el torneo y todos los partidos jugados hasta el momento.

2) **Seleccionar jugadores presentes**
   El asistente mostrará la lista de todos los jugadores y el administrador podrá marcar los que hayan asistido al evento.

3) **Generar partidos**
   Se generarán y se mostrarán en pantalla quince (como máximo) partidos a disputar entre todos los jugadores presentes de forma que se alternarán los jugadores entre un partido y el siguiente para que todos jueguen por igual. Además jugarán primero los que menos veces hayan jugado en la liga.

   Para cada partido se tienen que mostrar los nombres de los cuatro jugadores que intervienen agrupados por equipos (dos equipos).

4) **Anotar resultado de partido**
   Para cada partido que se muestra en pantalla se podrá anotar el resultado de cada equipo en un campo de texto que solo aceptará dígitos numéricos.

5) **Cambiar jugadores presentes y regenerar partidos (y volvemos al paso 4)**
   Esta función esta pensada para adaptar la aplicación a la variabilidad real en el conjunto de jugadores presentes durante la jornada. De esta forma los jugadores podrán irse o llegar más tarde al encuentro.

   Tras haber generado los partidos en incluso haber anotado algunos resultados de los partidos, se podrá volver al paso 2 para cambiar el conjunto de jugadores presentes. Se generarán entonces quince nuevos partidos de la misma forma que en el paso 3 teniendo en consideración los partidos que ya se han jugado en la jornada y mostrándolos al principio de la lista generada (recordando sus resultados).

6) **Guardar jornada en fichero SQL**
   Tras anotar los resultados de los partidos que se han jugado, se podrán guardar en un fichero que se escribirá en el sistema de ficheros del dispositivo móvil en formato de sentencias SQL (esto permite la opción de introducir los resultados directamente en el servidor de base de datos).

   Solo se guardarán los partidos en los que se haya anotado el resultado, que pueden ser solo algunos de los que se muestran y pueden estar alternados.

### 2.3.3 – Requisitos de eficiencia

Para garantizar la fluidez en la visualización de la web y minimizar el tiempo de carga de las distintas páginas de la aplicación, estas deberán ocupar el menor tamaño posible. Los archivos de imágenes deberán codificarse utilizando formatos adecuados que compriman la información sin que se aprecie pérdida en la calidad. Hay que prestar atención también al código de generación de las páginas, tendrá que estar bien optimizado para que el servidor tarde el menor tiempo posible en generarlas.

### 2.3.4 – Restricciones de diseño

El portal web se debe representar bien para cualquier resolución de pantalla, tomando como mínimo una resolución de 1024x768.

### 2.3.5 – Atributos

Para el desarrollo del portal web se deben usar tecnologías bien conocidas y que estén soportadas por los servidores de la mayoría de las empresas de *hosting* en Internet.

# 3 – Análisis

## 3.1 – Introducción

Tras la captación de requisitos viene la fase de análisis, en la que se busca un modelo para el sistema que queremos conseguir que satisfaga los requisitos. El objetivo de la fase de análisis es obtener un primer modelo conceptual que se irá ampliando y perfeccionando en las siguientes fases: nos servirá para poder diseñar y posteriormente implementar la aplicación.

Para modelar, igual que en la fase anterior, usaremos el formato que más se emplea en el ámbito de los proyectos software, la notación de UML (*Unified Modeling Language*). Es un lenguaje de propósito general para el modelado orientado a objetos. Nos permite representar el sistema mediante un conjunto de diagramas incluyendo tanto su estructura estática como su comportamiento dinámico.

Hemos visto ya en la especificación de requisitos que el sistema incluirá una pequeña herramienta en forma de asistente que llamamos GesPartidos y que además se ejecutará en una plataforma distinta. En la última sección de este capítulo se recoge el análisis de esta parte del sistema.

## 3.2 – Casos de uso

El diagrama de casos de uso muestra gráficamente toda la funcionalidad del sistema y lo hace desde el punto de vista de los usuarios. En el diagrama quedan representados como actores todos los usuarios o entidades externas al sistema que interaccionan con el mismo. Los casos de uso representan los requisitos funcionales de la aplicación y cada uno se asocia a los actores que intervienen en él.

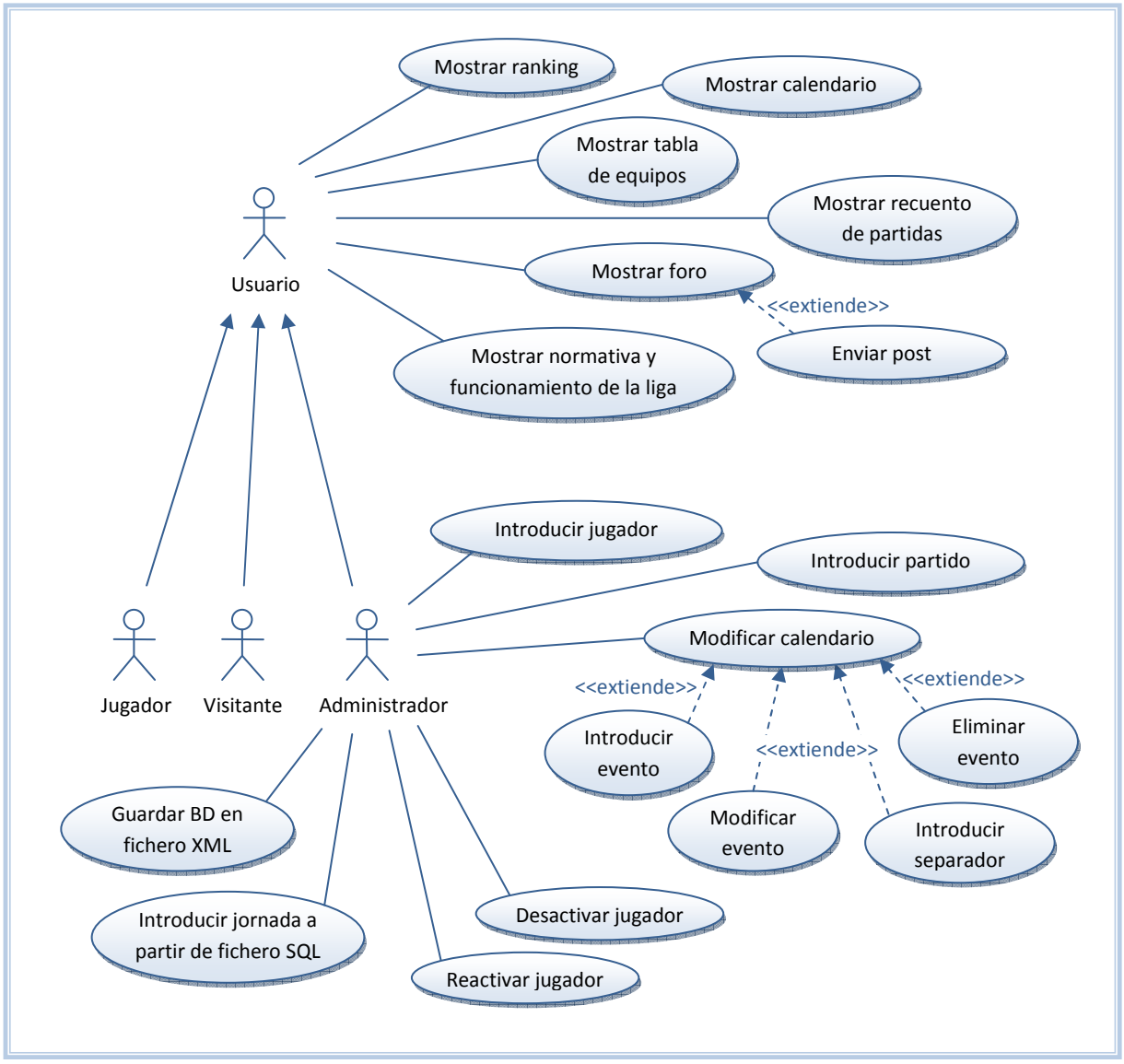


*Figura 5: Diagrama de casos de uso*

En el diagrama vemos que el actor *Usuario*, que representa cualquier usuario del sistema, interviene en todos los casos de uso en que se visualiza información de la liga y además puede introducir mensajes en el foro.

Del actor *Usuario* heredan los tres tipos de usuarios que tiene nuestro sistema. El *Administrador*, además de lo anterior, realiza otras tareas en el sistema destinadas a la gestión de la liga.

## 3.3 – Diagrama de clases

Mediante el diagrama de clases describimos la estructura estática de nuestro sistema. En este diagrama se representa el conjunto de entidades (tipos o clases de objeto) que componen el sistema y cómo están relacionadas entre sí.

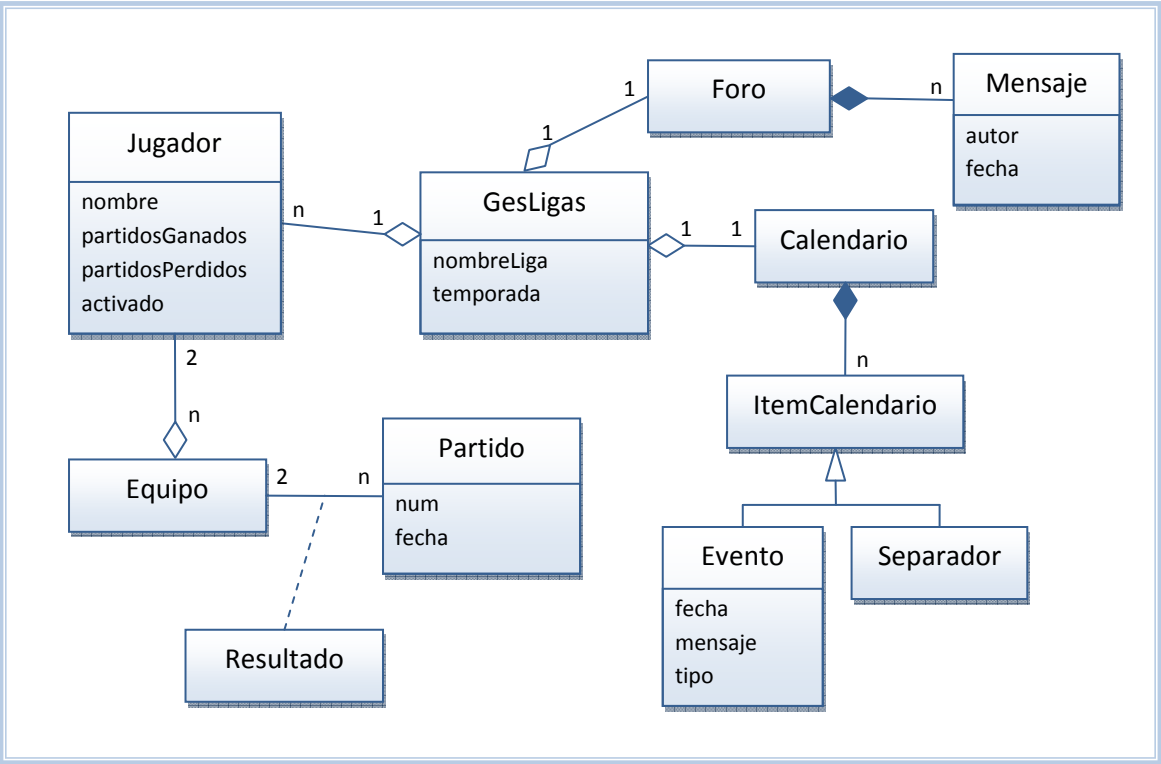


*Figura 6: Diagrama de clases*

En el diagrama podemos ver que la entidad principal y que se encuentra en la parte central se llama *GesLigas* y de ella parten el resto de entidades del diagrama. *GesLigas* representa la liga que se esta gestionando y contiene como atributos las características generales que definen la liga como su nombre o a qué temporada pertenece.

La liga contiene un conjunto de varios jugadores que tienen un nombre para identificarse, pueden estar activados o no y para cada uno se lleva la cuenta de partidos ganados y perdidos. El resto de información que se demanda de un jugador se deriva de estos atributos.

Cada jugador puede formar parte de varios equipos, formará un equipo diferente con cada uno del resto de jugadores. Cada equipo esta formado por dos jugadores. Por otra parte cada equipo puede haber jugado en varios partidos y en cada partido intervienen dos equipos. Cuando un equipo participa en un partido obtiene un resultado en ese partido.

La Liga se compone también de un foro y un calendario. Estas dos entidades son parecidas a nivel conceptual: el foro es una lista de mensajes y el calendario es una lista de eventos. Sin embargo hay que incluir en el calendario los separadores. En fases posteriores esto puede cambiar, pero en esta fase de análisis se distinguen los eventos de los separadores pues conceptualmente son cosas distintas.

## 3.4 – Análisis de GesPartidos

### 3.4.1 – Casos de uso

GesPartidos es un asistente destinado a un solo caso de uso que solo realizará el administrador.

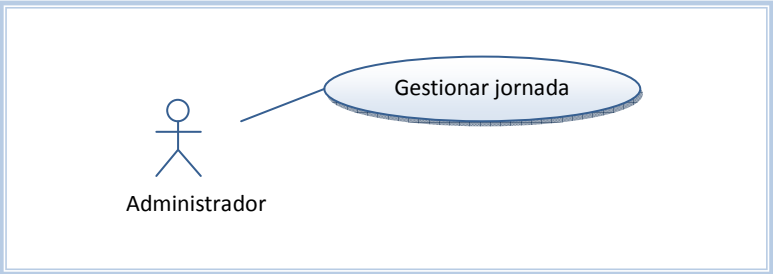


*Figura 7: Diagrama de casos de uso de GesPartidos*

Este caso de uso conviene detallarlo y lo haremos mediante la típica plantilla que se emplea para esto y que describe la sucesión de eventos que se producen.

| | |
|---|---|
| **Caso de uso** | Gestionar jornada |
| **Actores** | Administrador (iniciador) |
| **Propósito** | Gestionar los partidos en pista |
| **Resumen** | Se marcan los jugadores asistentes a la jornada y de acuerdo a esta información y el histórico de partidos se genera la lista de partidos. A continuación se recogen los resultados y se guardan en un fichero. |
| **Precondiciones** | Existe (y es accesible) un fichero en formato XML que contiene los datos de todos los jugadores y el histórico de partidos. |
| **Postcondiciones** | Los resultados quedan almacenados en un fichero en formato de sentencias SQL de actualización de la base de datos. |
| **Incluye** | - |
| **Extiende** | - |
| **Hereda de** | - |

| **Intenciones de usuario** | **Obligaciones del sistema** |
|---|---|
| 1. El administrador inicia el asistente. | 2. El sistema muestra una lista de todos los jugadores registrados en la liga. |
| 3. El administrador selecciona los jugadores que han asistido al evento. | 4. El sistema genera quince (como máximo) partidos a disputar entre los jugadores presentes de forma que se alternarán los jugadores entre un partido y el siguiente para que todos jueguen por igual. Además jugarán primero los que menos veces hayan jugado en la liga. |

| | |
|---|---|
| | Los partidos se muestran en pantalla indicando para cada uno los nombres de los jugadores que intervienen en cada equipo. Se muestra además para cada partido una casilla para apuntar los resultados de cada equipo. |
| 5. El administrador apunta los resultados de cada partido. Cuando termina, solicita guardar los resultados. | 6. El sistema guarda todos los partidos en los que se haya anotado el resultado en un fichero con las sentencias SQL necesarias para actualizar la base de datos. |
| **Extensiones síncronas** | |
| #1. En el paso 5 el administrador podrá solicitar cambiar los jugadores, en ese caso, el sistema volverá al paso 2 sin perder los partidos en los que ya se han apuntado resultados y en el paso 4 los tendrá en cuenta como partidos que ya se han jugado. | |
| **Extensiones asíncronas** | |
| #1. En cualquier momento el administrador puede salir del asistente cancelando toda la gestión y por consiguiente no se generará ningún fichero de resultados. | |

### 3.4.2 – Diagrama de clases

El diagrama de clases para GesPartidos es una parte del diagrama de clases de GesLigas pues, siendo el mismo sistema se encarga de gestionar una parte en la que intervienen solo las entidades de *GesLigas* (que aquí renombramos a *GesPartidos*), *Jugador*, *Equipo*, *Partido* y *Resultado*.

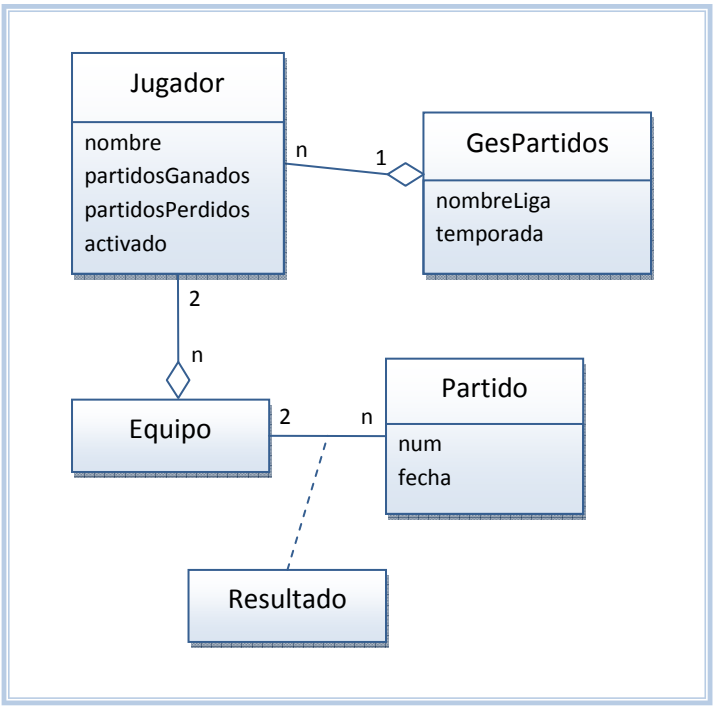


*Figura 8: Diagrama de clases de GesPartidos*

# 4 – Diseño

## 4.1 – Arquitectura del sistema

La mejor forma de diseñar un sistema generalmente es separarlo en capas bien delimitadas y que se comunican entre ellas (cada una con la inmediatamente superior y con la que tiene inmediatamente por debajo) a través de una interfaz (también llamada API). Esta forma de trabajar obliga en cierta forma a organizar y estructurar bien el sistema, permite separar las tareas de implementación, facilitando así además la paralelización del trabajo en un equipo de desarrollo. Incluso se podrá sustituir fácilmente en una capa una implementación por otra totalmente distinta mientras se cumpla con la interfaz. Por otra parte, la separación entre capas puede ser tal que cada capa resida en distintos ordenadores, como ocurre en nuestro caso.

Usaremos la "arquitectura en tres capas" que además es lo que suele hacerse en la mayoría de aplicaciones informáticas. A continuación describimos cada una de ellas por orden:

- **Capa de presentación**. Se le llama también Interfaz Gráfica de Usuario (IGU). Interactúa con el usuario recogiendo las acciones que realiza (mediante el teclado, ratón o cualquier periférico de entrada), haciendo en ocasiones una primera verificación de los datos introducidos, enviando los datos a la capa inferior y mostrando adecuadamente al usuario los resultados que recibe de esta.
- **Capa de lógica de negocio** (o simplemente capa de negocio). Recibe las peticiones, comandos o datos de entrada de la capa de presentación comprobando su corrección, coherencia y/o validez; se procesan los datos y se envían las respuestas correspondientes a al capa de presentación. En ocasiones, en el proceso se requiere almacenar datos de forma permanente o recuperar datos guardados previamente; de esto se encarga la capa de persistencia.
- **Capa de persistencia de datos**. Es donde residen los datos que requieren ser almacenados por la aplicación. Los datos se organizan en bases de datos. Esta capa está formada por uno o más gestores de bases de datos que se encargan de acceder a los dispositivos de almacenamiento masivo del sistema para añadir, modificar o borrar los datos de la aplicación.

Frecuentemente se usan los términos *capa* y *nivel* indistintamente, sin embargo, puede hablarse de *niveles* según la forma en que las *capas* se distribuyen de forma física. Por ejemplo, en una solución de tres *capas* que residen en un solo ordenador, se dice que la arquitectura es de tres *capas* y un solo *nivel*. En nuestro caso habrá tres *capas* alojadas en tres *niveles*, como podemos ver en la figura 9, y que se desarrollan en los siguientes apartados.

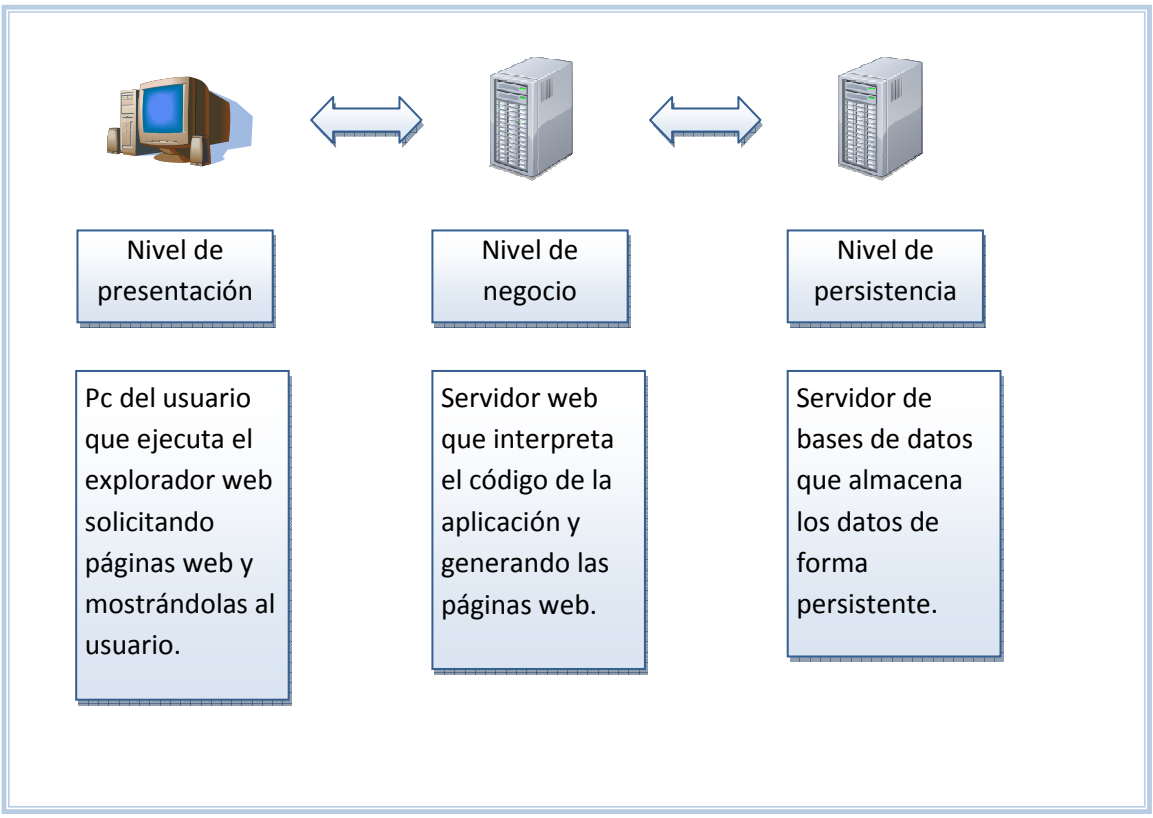


*Figura 9: Tres capas - tres niveles*

## 4.2 – Capa de presentación

Nuestra aplicación dispondrá de una interfaz web por lo que la capa de presentación está claramente diferenciada y se ejecutará en cada uno de los ordenadores de los usuarios mostrándose dentro de un explorador.

La interfaz que usaremos para la comunicación con el nivel de negocio se define mediante el estándar de código XHTML para el contenido de las páginas web y el protocolo HTTP para la transferencia de estas a través de la red. Todos los exploradores web conocen HTTP y saben interpretar el lenguaje XHTML, en consecuencia podrán representar en pantalla nuestra interfaz gráfica de usuario.

### 4.2.1 – Diseñando la interfaz

La interfaz será simple y atractiva siguiendo un estilo moderno (con objetos redondeados, sombras, transparencias) pues el objetivo es crear afición para la liga. Debe ser fácil y rápido acceder a la información que se requiera conocer en cada momento.

En todas las páginas del portal se va a seguir un formato unificado tanto a nivel estructural como a nivel de estilo (fuentes, líneas, tablas, etc.)

La estructura que seguirán todas las páginas se representa mediante el esquema que se muestra en la figura 10.

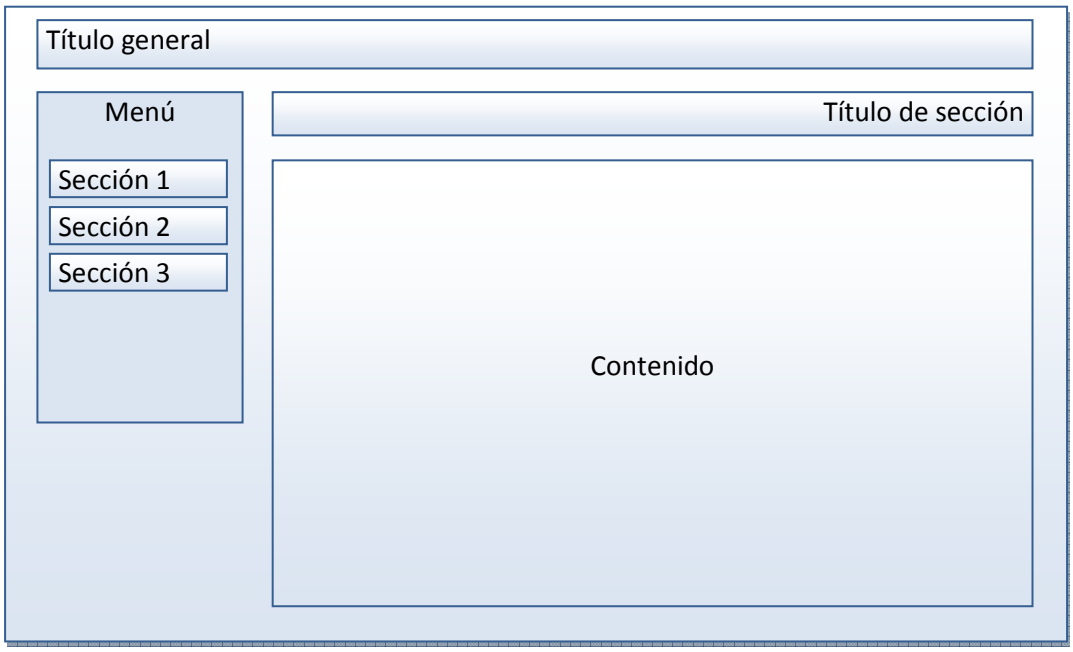


*Figura 10: Esquema de la estructura general de la interfaz*

La aplicación práctica inmediata para este sistema de gestión de ligas es una liguilla de frontenis que nos servirá además para la fase de pruebas. Es por ello que vamos a enfocar en este sentido el diseño de la interfaz y el contenido de ciertas secciones (por ejemplo la sección

de normativa). Esto no significa que vaya a ser exclusivamente para frontenis, pues el funcionamiento es el mismo para los otros deportes, tan solo habría que cambiar pequeños detalles de la interfaz. De hecho se puede ir ampliando a otros deportes a medida que se requiera formando una colección de “temas” para la interfaz (por ejemplo un fondo distinto para cada deporte.)

Estos “temas” describirían el estilo visual de la web que, como se ha dicho antes, será el mismo en todas las páginas. En este caso vamos a poner un fondo acorde al frontenis, como lo es por ejemplo, una foto de una pista de frontón. Y acorde al fondo se usarán distintos tonos de verde para el texto, títulos, líneas, tablas, etc.

### 4.2.2 – Interfaz de la parte pública

Habrá una página por cada sección a la que se accederá con cada entrada del menú. En todas las páginas se mostrará el menú y por tanto en cualquier momento y desde cualquier sección se podrá acceder a las demás.

A continuación vamos a ver el diseño de las principales secciones.

## Ranking

Esta será la página de entrada al portal web. Es la información más requerida por todos los tipos de usuario y además así fomentamos la competitividad, necesaria en cualquier torneo.

Se muestra la puntuación de todos los jugadores incluyendo los que están desactivados que aparecen tachados. Además es importante que estén ordenados según el número de puntos que tiene cada jugador (primero el que más puntos tiene).


*Figura 11: Diseño de la interfaz del ranking*

## Calendario

En la sección de calendario se mostrará una lista de eventos en el orden en que el administrador los introdujo en el sistema (primero el más reciente). En el mensaje de los eventos podrá mostrarse, además de texto, también imágenes, tablas de resultados de la jornada y en definitiva cualquier cosa que pueda representarse mediante código XHTML. El contenido de los mensajes se mostrará además siguiendo el estilo general del portal web.

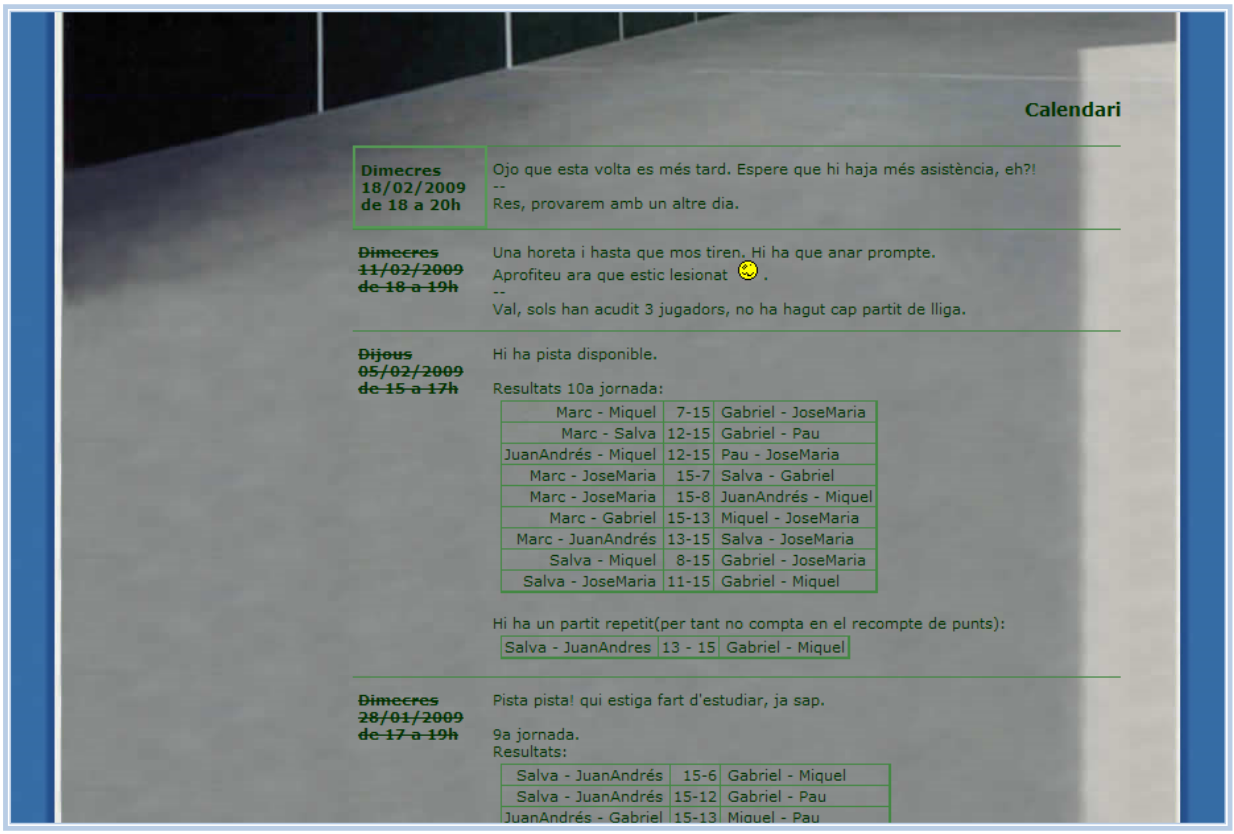


*Figura 12: Diseño de la interfaz del calendario*

## Chat-foro

El chat-foro es una lista de mensajes de los usuarios. La lista se divide en páginas cuando es muy larga y se puede acceder a todas las páginas. En cada mensaje aparece, además del texto del mensaje, el nombre del autor y la fecha en que se escribió. Cualquier usuario podrá añadir un mensaje rellenando los campos de “Tu nick” y “Tu mensaje” y presionando sobre el botón de enviar. En el contenido del mensaje pueden incluirse emoticonos predefinidos escribiendo su representación en texto o eligiendo uno de ellos desde el desplegable. En cuanto se envía un mensaje se actualizará la página y aparecerá en la lista (siempre que pase la validación).

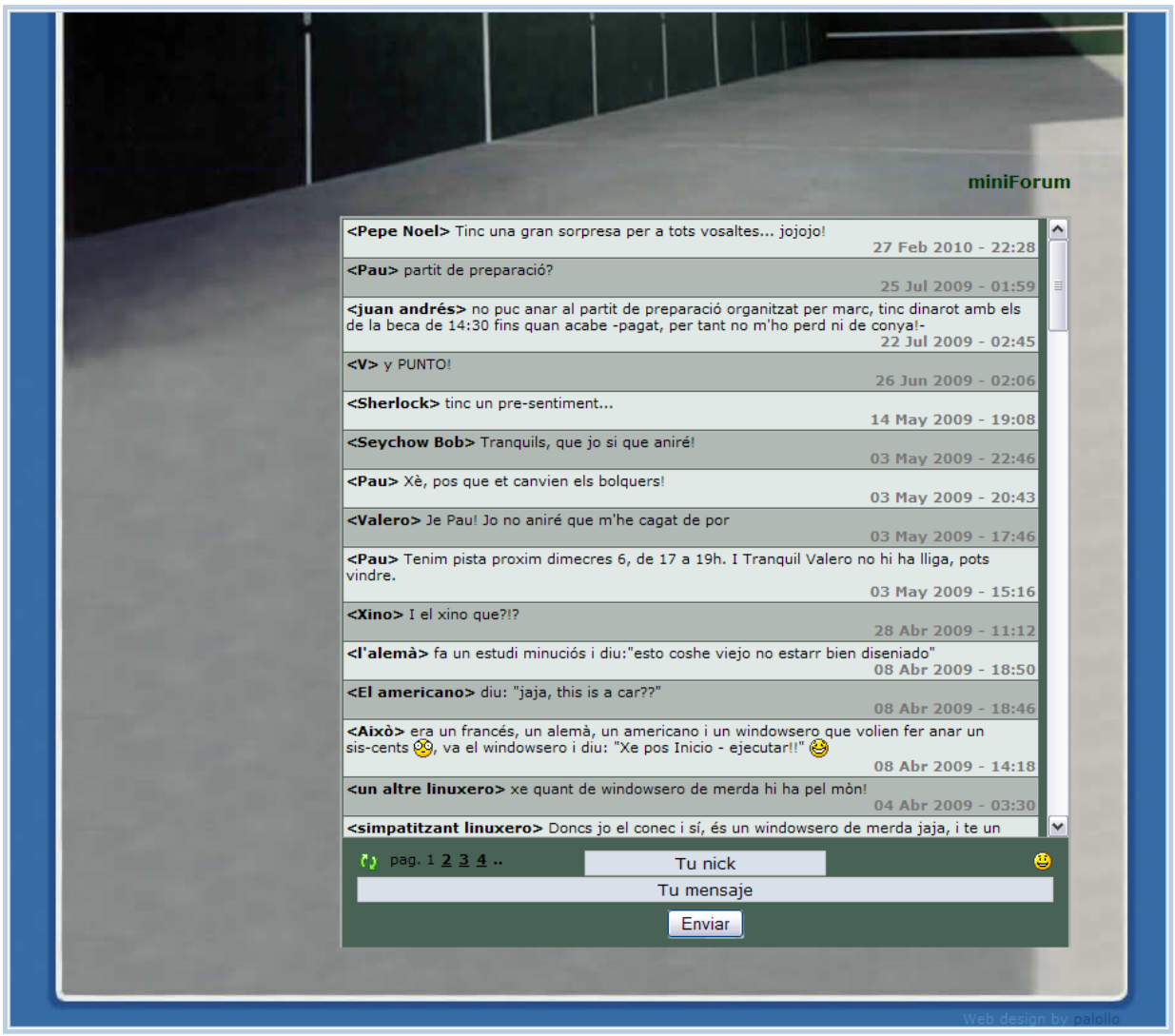


*Figura 13: Diseño de la interfaz del foro*

### 4.2.3 – Interfaz de la parte privada

La parte privada sigue el mismo esquema de página que incluye el menú y por lo tanto podremos acceder en todo momento a cualquiera de las secciones de la parte pública para ver el resultado de la gestión. La página general de administración y punto de entrada a la parte privada se muestra en la figura 14.

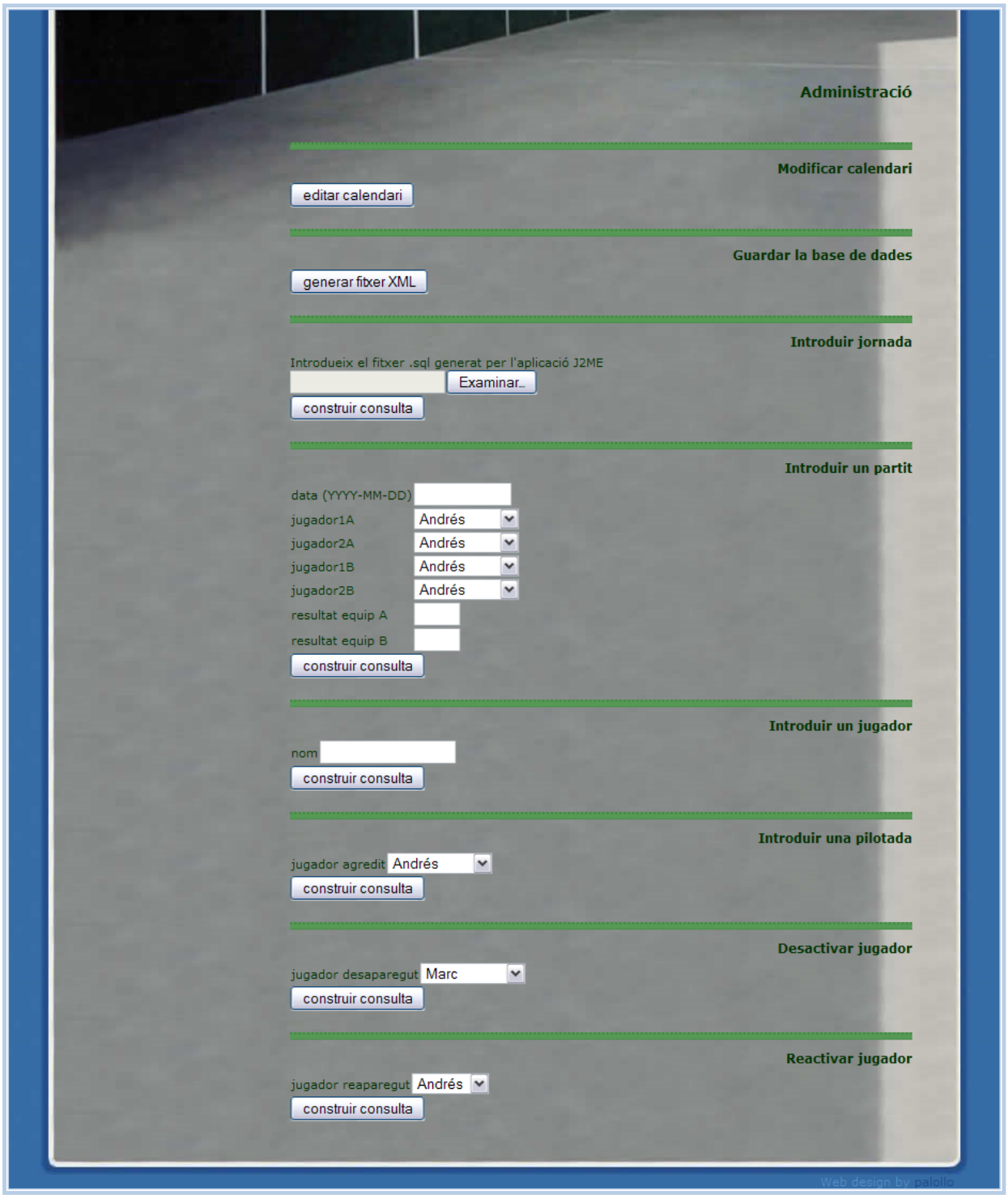


*Figura 14: Diseño de la interfaz de la página general de administración*

Desde aquí se pueden realizar todas las tareas de gestión que permite la aplicación. La mayoría se pueden realizar directamente en esta página. La funcionalidad de cada una de ellas se detalla en el próximo apartado *4.3 – Capa de lógica de negocio*.

Las gestiones sobre el calendario se realizan en la página de edición del calendario a la que se accede a través del botón “editar calendari”. Mediante el botón “generar fitxer XML” accederemos a la página de descarga de la base de datos para la aplicación móvil GesPartidos. Y en el resto de tareas se deben introducir los datos correspondientes y, tras pulsar sobre el botón “construir consulta” accedemos a la página de previsualización de la consulta.

En la figura 15 se muestra un diagrama de navegabilidad que recoge todas las páginas accesibles de la parte privada.

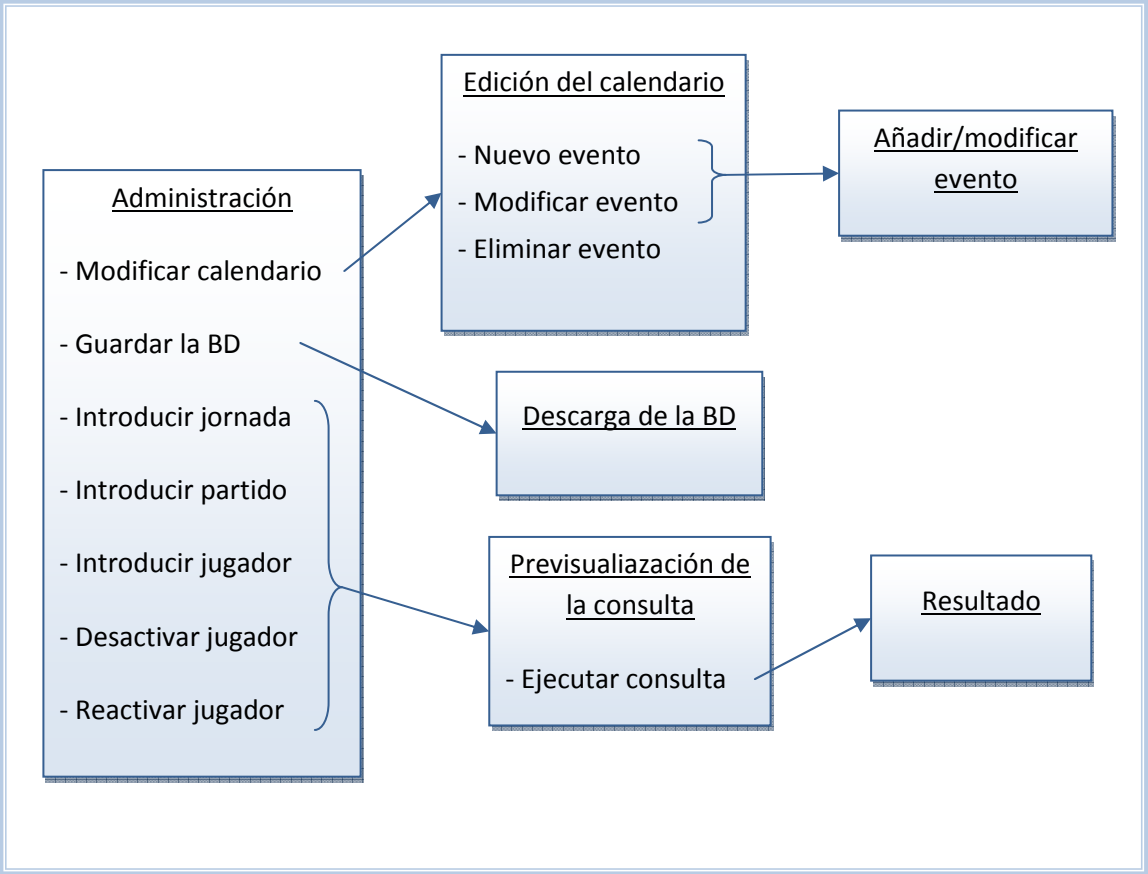


*Figura 15: Esquema de navegación de la parte privada*

Seguidamente se describen el resto de páginas que intervienen en el diagrama.

## Página de edición del calendario

Es igual que la página de calendario de la parte pública, de forma que vemos siempre el resultado de la edición, pero además incluye un botón al principio para añadir nuevos eventos y sendos botones para modificar y eliminar en cada evento.

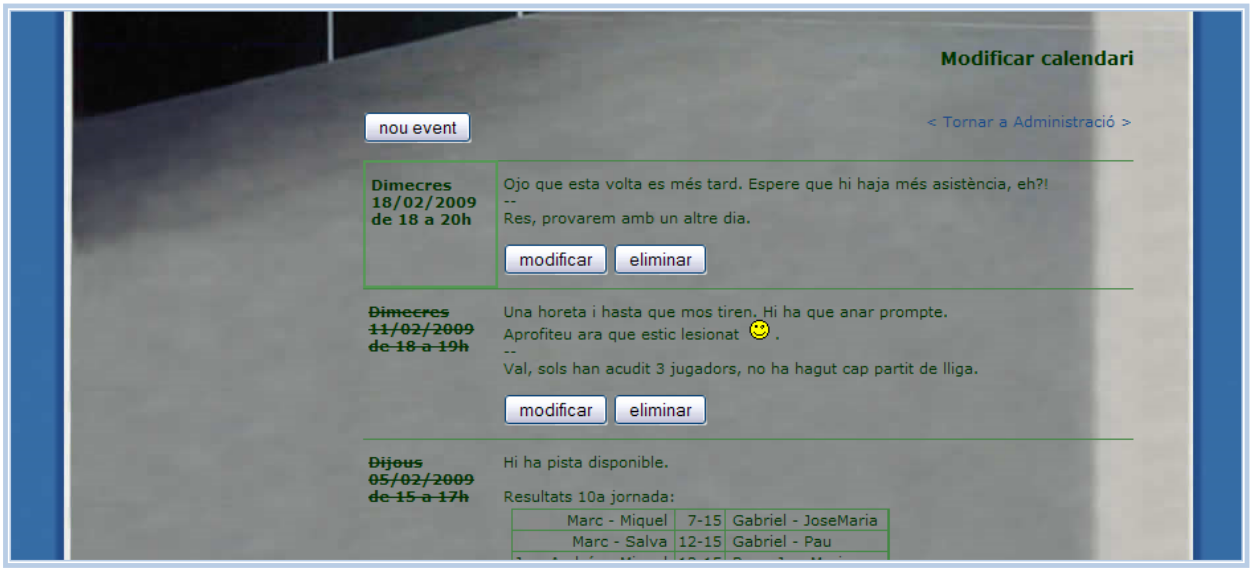


*Figura 16: Diseño de la interfaz de edición del calendario*

## Nuevo evento / Modificar evento

Al pulsar sobre el botón para añadir eventos o en alguno de los botones de modificar se abrirá una ventana del explorador aparte donde se visualizará el código XHTML de la fecha y del mensaje (en el mensaje además se pueden insertar componentes predefinidos como una tabla de resultados de una jornada como se ve en la imagen inferior). Podremos editar los campos y elegir el tipo de evento. Tras pulsar en “enviar al servidor”, se actualizará la página de edición del calendario para ver el resultado.

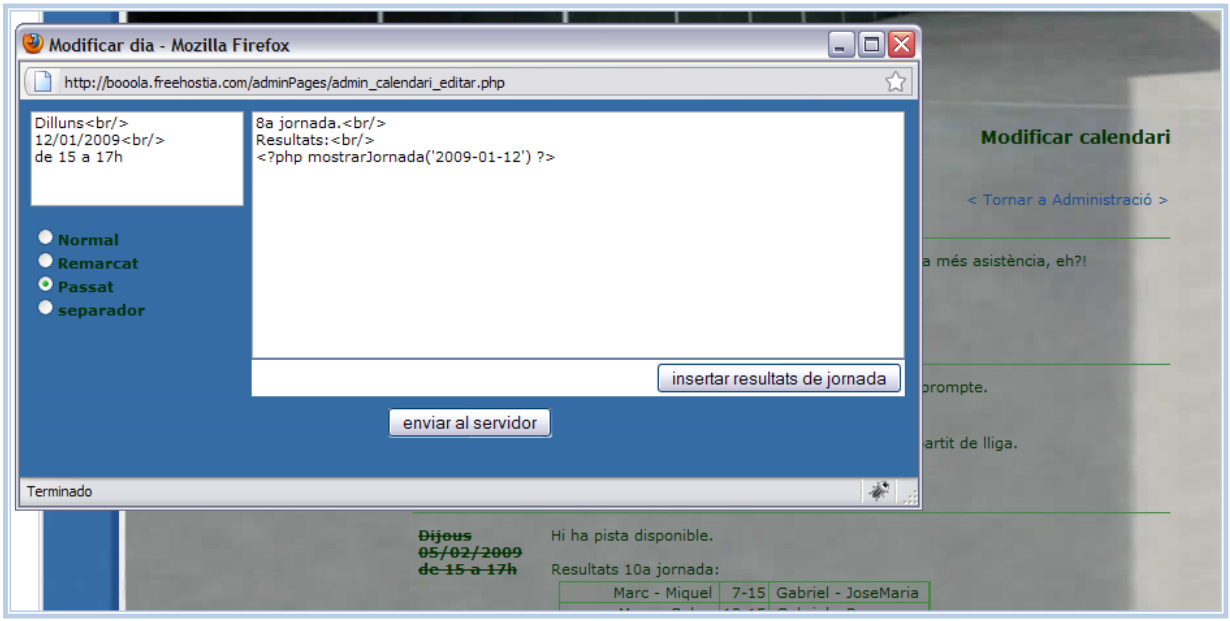


*Figura 17: Diseño de la interfaz para modificar un evento*

## Descarga de la BD

Desde la página principal de administración de la liga podemos acceder a esta sección que nos ofrece un enlace mediante el cual podremos descargar un fichero donde se ha guardado la base de datos del portal web y que nos sirve como entrada para el asistente de GesPartidos y también como copia de seguridad.

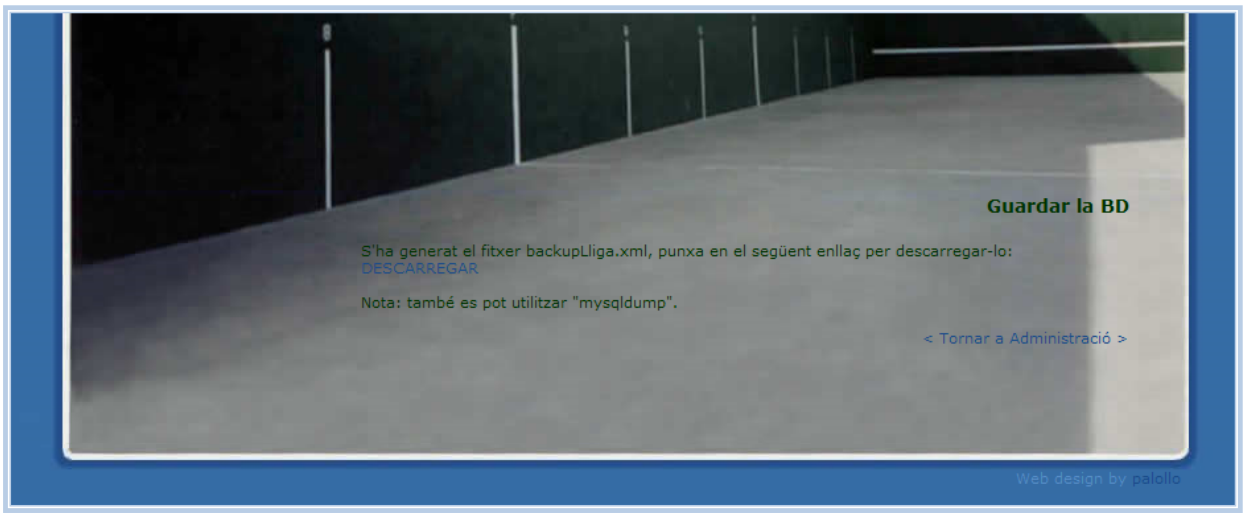


*Figura 18: Diseño de la interfaz de la página de descarga de la base de datos*

## Previsualización de la consulta

La mayoría de las tareas de gestión de la liga implican modificar algún campo de la base de datos. Esta página representa un paso intermedio justo antes de ejecutar la transacción sobre la base de datos. Aquí podemos ver todas las operaciones que modificarán nuestra base de datos en lenguaje SQL. Con este paso facilitamos la implementación, el mantenimiento y la vez aportamos una función didáctica a la aplicación que viene bien siendo un PFC. El administrador que no entienda el lenguaje o simplemente no necesite saber esta información, puede ignorarla y pulsar directamente sobre el botón “executar”.


*Figura 19: Diseño de la interfaz de previsualización de la consulta*

## Resultados

Tras realizar cualquier tarea de gestión que modifique de algún modo la base de datos se mostrarán, en una ventana aparte del explorador, los resultados que devuelve el gestor de bases de datos para cada operación que se le haya enviado.

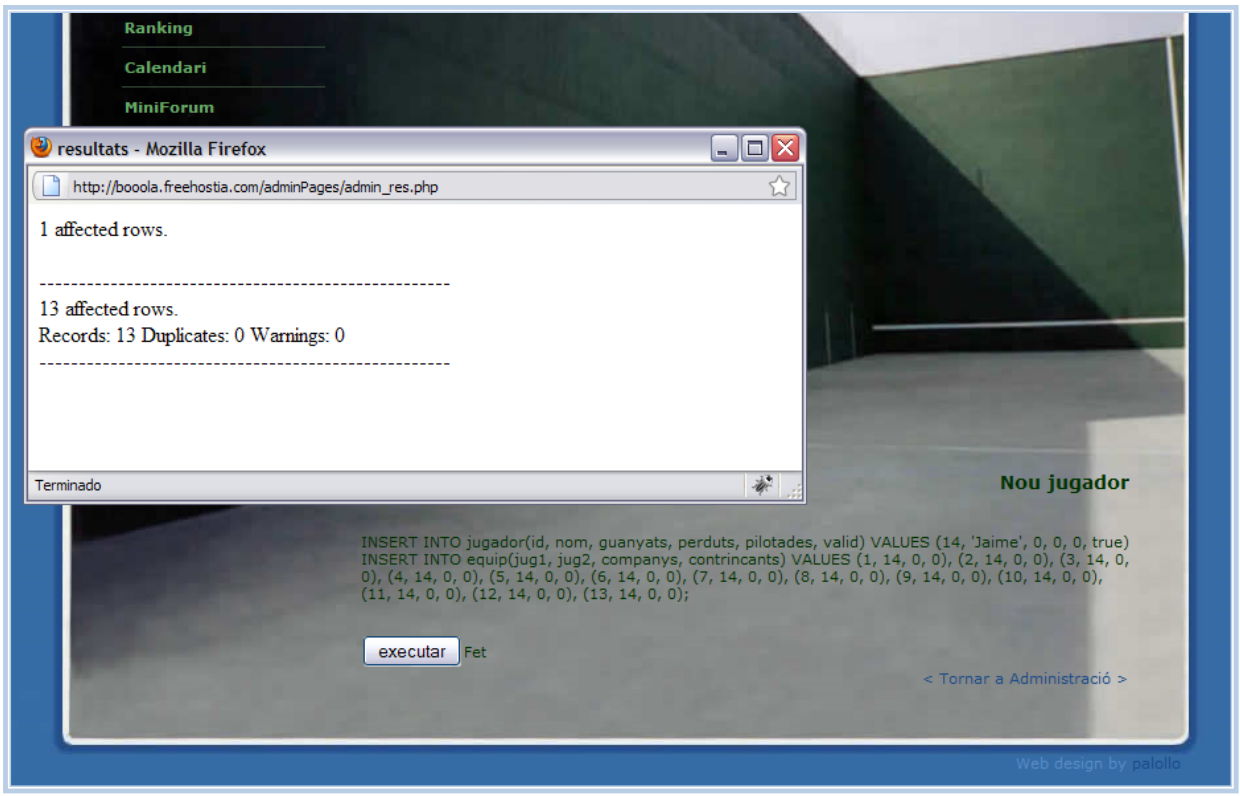


*Figura 20: Diseño de la interfaz de resultados de la consulta*

## 4.3 – Capa de lógica de negocio

La capa de lógica de negocio se ejecutará en una máquina que actuará como servidor web. El servidor web se encargará de interpretar el código de la aplicación, solicitando a la capa de persistencia los datos que se requieran en cada operación, generando las páginas web y enviándolas por HTTP al explorador cliente en la máquina del usuario.

La lógica varía en cada página, pero generalmente se hace lo siguiente:

- En **páginas que muestran información** (como la de ranking o la del calendario) se conecta con la base de datos y se realizan consultas sobre las tablas que contienen la información que debe mostrarse, almacenándose en memoria y completando con ella el contenido de la página web que se envía al cliente.

- En **páginas de introducción de datos** (como la página principal de administración o la del foro) se accede también a la base de datos para consultar la información que se tenga que mostrar, por ejemplo para rellenar las listas desplegables en donde el usuario deberá elegir un jugador de entre los que ya se han registrado. Entonces se genera el formulario y se envía al navegador. Conforme el usuario introduce los datos el mismo navegador (capa de presentación) hará una validación previa de los datos. Cuando el usuario confirma la operación se solicita al servidor la página de resultados enviándole los datos que se han introducido.

- En **páginas de resultados** se reciben los datos introducidos por el usuario (si los hay), se accede a la base de datos para realizar las operaciones de inserción, actualización o borrado en las tablas que corresponda y se muestran los resultados que devuelve el gestor de bases de datos.

En todas las páginas se emplean funciones para generar la cabecera y el pie de la página.

A continuación vamos a detallar la lógica de algunas de las páginas características en este aspecto.

### Ranking

Este es un ejemplo de página que solo muestra información. El proceso de generación de la página es el siguiente:

- Se llama a la función que genera la cabecera con el menú.
- Se conecta con la base de datos y se solicitan todos los registros de la tabla que almacena la información de jugadores.
- Se genera la tabla de ranking con la información de los jugadores.
- Si hay algún jugador desactivado se muestra la leyenda.
- Por último se llama a la función que genera el pie de página.

### Chat-foro

Esta página permite visualizar los mensajes que se han ido introduciendo y a la vez permite introducir nuevos mensajes.

En la visualización de los mensajes se siguen las pautas generales para *páginas que muestran información*. Sin embargo hay que destacar lo siguiente:

- El foro se divide en páginas cuando el número de mensajes sobrepasa un valor configurable. Por tanto, al solicitar a la base de datos los registros de la tabla donde se almacenan los mensajes, solo se piden los que corresponden a la página que se visualizará.
- La página a visualizar inicialmente es la primera. En la parte inferior se incluyen enlaces a las siguientes tres páginas y a las tres páginas anteriores. Al pulsar sobre alguno de estos enlaces se envía al servidor el número de página solicitada usando el método GET del protocolo HTTP. Esto significa que la información se incluye como un parámetro en la URL de forma que, por ejemplo, para acceder directamente a la quinta página se podría introducir en el explorador la dirección: *www.miservidorweb.com/publicPages/foro.php**?pag=5***
- Para cada mensaje se cortan las palabras que excedan un máximo de caracteres configurable. Esto se hace porque una palabra tiene que estar entera en la misma línea, con lo que una palabra muy larga obligaría a tener una línea a su vez muy larga que se saldría del contenedor o (en caso de que el contenedor tenga establecido el ancho automático) ensancharía el contenedor descuadrando el diseño de la página.
- En cada mensaje hay que sustituir las cadenas de caracteres que simbolizan un emoticono por el código que muestre la imagen correspondiente.

Tras la lista de mensajes se muestra un pequeño formulario que permite introducir nuevos mensajes. En esta parte vamos a destacar algunos detalles en orden cronológico:

- El servidor genera el formulario junto con el resto de la web y lo envía al explorador cliente.
- En el explorador web del usuario se ejecutan una serie de funciones que permiten la interacción con el usuario. Hay una función para validar el contenido de las entradas, otra para limpiar el contenido u otra para insertar emoticonos en el campo del mensaje desde el desplegable. Estas funciones forman parte de la lógica de la aplicación, sin embargo se ubican en el nivel de presentación ya que intervienen en la interacción con el usuario y esto solo lo puede hacer la capa de presentación.
- Cuando el usuario pulsa sobre el botón *Enviar*, el navegador solicita al servidor la página de resultados enviando, junto con la petición, los datos introducidos en los campos del formulario usando el método POST del protocolo HTTP.
- Siempre que el servidor recibe una petición de la página de resultados del foro espera tener como datos adjuntos un *nick* y un *mensaje*. Entonces valida los datos (una segunda validación necesaria por seguridad, pues no hay garantías de que se ejecute siempre la primera) y a continuación los procesa: se eliminan ciertos caracteres que se añaden en el envío y, los que coinciden con caracteres de control de alguno de los lenguajes que se emplean, se escapan o se transforman en cadenas equivalentes.
- Finalmente se inserta en la base de datos el nuevo mensaje con la fecha actual del servidor web (más un desfase configurable) y se envía al navegador cliente la página del foro actualizada.

### Administración

En la página general de administración se generan muchos formularios pequeños, uno para cada operación. Para rellenar las listas desplegables de jugadores hay que acceder a la base de datos para obtener los jugadores (activados o desactivados según el caso) y sus identificadores.

Ya en el navegador del usuario, los datos que requieren validación se validan en cuanto el cursor abandona el campo. En todos los formularios hay un botón de tipo *submit* que cuando se presiona se solicita al servidor la página de *previsualización de la consulta* enviándole los datos del formulario correspondiente (si los hay) por el método POST (en el cuerpo de la petición HTTP) y además se le indica la operación a realizar usando el método GET (en la dirección URL).

De nuevo en el servidor se construyen una o varias consultas en SQL según la operación solicitada y con los datos que se le pasan e incluso, en algún caso, accediendo de nuevo a la base de datos para recuperar información necesaria para el proceso. Entonces se envía la página de *previsualización de la consulta* al navegador. En la página se incluye un formulario con campos ocultos (uno para cada consulta) que solo sirven para propagar las consultas que acaba de generar el servidor y que se le enviarán de nuevo en cuanto el usuario pulse sobre el botón “executar”. Esto puede parecer redundante, sin embargo es necesario pues el servidor no mantiene información entre peticiones, funciona así (a menos que se usen sesiones, pero en este caso es más sencillo de esta forma).

En el siguiente paso solo queda ejecutar las consultas sobre la base de datos y enviar al usuario las respuestas obtenidas para que sepa si el resultado de la operación fue satisfactorio.

### Calendario y Edición del calendario

Para generar estas páginas, como se viene haciendo en los casos anteriores, de nuevo se accede a la base de datos para recuperar todos los registros de la tabla donde se almacenan, en este caso, los eventos del calendario. Un detalle a tener en cuenta es que los campos de fecha y mensaje realmente son código XHTML que debe interpretarse y mostrarse adecuadamente. Esto es tarea del navegador web del usuario y por tanto solo hay que insertar el contenido de estos campos “tal cual” en la lista de eventos del calendario y el navegador ya se encarga de interpretar el código junto con el resto de la web. Además se pueden mostrar resultados de jornadas. Esto otro ya no es tan simple. Para conseguirlo, el servidor debe reconocer llamadas a funciones dentro del campo del mensaje, se explica en el apartado *5.3 - Detalles de implementación*.

Hasta aquí, la página de visualización del calendario de la parte pública y la de su edición en la parte de administración son iguales, pero además la página de edición incorpora unos controles para editar los eventos. En este caso se crea un solo gran formulario con muchos botones: uno al principio para añadir eventos, y dos por cada evento existente (el de modificar y el de eliminar). Según el que se pulse se asignará valor a dos campos ocultos del formulario: uno para la operación (nuevo, modificar o eliminar) y el otro para el número del evento a modificar o eliminar.

Cuando el usuario pulsa sobre alguno de los botones se envía el formulario y el navegador abre una pequeña ventana secundaria pero no modal donde mostrará la página con que responda el servidor. Este último, al recibir el formulario y según el campo de operación, accederá a la base de datos para recuperar el evento, si se quiere que modificar, o eliminarlo, si lo que se solicita es su borrado. En las operaciones de añadir y modificar se enviará el formulario de edición con los campos vacios o rellenándolos respectivamente. En la operación de borrado se enviará la respuesta del gestor de base de datos como confirmación de la operación.

Cuando se muestra el formulario de edición de un evento, aparece un botón al final para confirmar los datos enviándolos al servidor, que actualizará la base de datos y devolverá la respuesta del gestor de bases de datos, igual que en el borrado.

## 4.4 – Capa de persistencia de datos

La capa de persistencia se compone de un servidor de bases de los datos, que puede ejecutarse en la misma máquina que el servidor web u otra distinta (con lo que tendríamos el tercer "nivel"), y de un conjunto de ficheros de recursos (imágenes, videos, ficheros de texto y otros) a los que accede el servidor web en algunos casos para consultar datos y en otros para enviarlos junto con las páginas web.

La base de datos se organiza en tablas que pueden relacionarse entre sí. Para representar su estructura se utiliza el diagrama entidad-relación que veremos a continuación. Es similar al diagrama de clases que usábamos en el capítulo de Análisis, pues representa también entidades y relaciones entre estas. Sin embargo antes hablábamos de "clases" como modelos de entidades del mundo real que se incluyen en nuestra aplicación y que se encuentran, en algún momento, en memoria; y ahora hablamos de "tablas" como modelos de entidades reales que se van a guardar de forma persistente en una base de datos (en disco). Lo que antes eran "objetos" como instancias de una clase, ahora son filas o "registros" de una tabla. Y lo que antes eran "atributos" de las clases, ahora son las diferentes columnas o "campos" dentro de las tablas que en definitiva son propiedades de la entidad real.

Lógicamente para guardar en disco uno de estos "modelos" antes suele pasar por memoria. Es decir, lo normal es guardar ciertos objetos con los que trabaja la aplicación. Por tanto, hay una cierta relación (de inclusión) entre las entidades representadas en el diagrama de clases y las que se encuentran en el diagrama entidad-relación.

Sin embargo no todas las clases se guardan y no todos sus atributos se incluyen. Por tanto no tiene que coincidir cada tabla con una clase, la estructura puede variar ya sea por requerimiento del tipo de base de datos que empleemos, por cuestión de eficiencia o por otros motivos, pues en definitiva el ámbito es distinto.

### 4.4.1 – Diagrama entidad-relación

En el diagrama que se muestra a continuación se representan las tablas como recuadros, las relaciones entre tablas como rombos y los campos de cada tabla como elipses a su alrededor.

Se parece en cierto modo al diagrama del apartado *3.3 – Diagrama de clases* pero cambian algunos detalles por las razones que hemos explicado antes y, además, nos encontramos en una fase posterior en el desarrollo: lo que en la fase de Análisis era una idea conceptual de las entidades que iban a formar parte de nuestra aplicación, ahora son tablas que vamos a crear en la base de datos. Por ejemplo, las relaciones entre *Jugador*, *Equipo* y *Partido* se han simplificado: la tabla *Equipo* solo se accede para la generar la página que muestra la tabla de equipos y de esta forma cuando se muestran los partidos con los nombres de los jugadores se accede directamente a la tabla *Jugador* sin pasar por la de *Equipo*. Con esto se mejora el tiempo de respuesta empleando un poco más de espacio en disco.

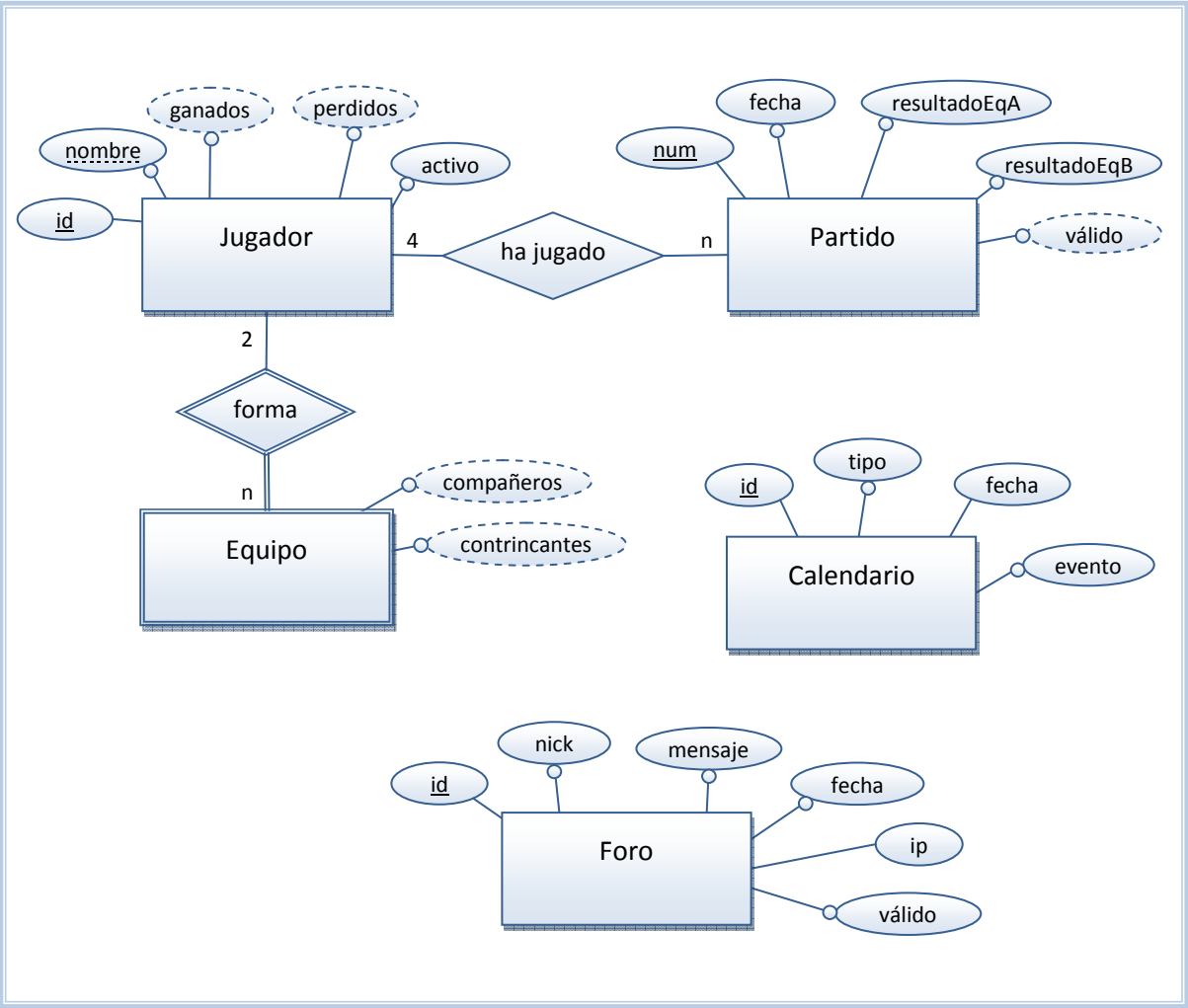


*Figura 21: Diagrama entidad-relación*

Se guardarán, en la tabla *jugador*, todos los jugadores registrados en la liga, con un identificador único para cada uno, un nombre y un booleano que indicará si está activo o no.

Cada jugador puede haber jugado muchos partidos. En cada partido deben haber jugado cuatro jugadores. Los partidos se almacenan en la tabla *Partido*, se identifican con un número, y de ellos se guarda la fecha en que se desarrollaron, los resultados de cada equipo (forman equipo el primero con el segundo y el tercero con el cuarto) y un booleano para saber si es válido (todos los jugadores que intervienen están activos) o no.

En la tabla *Equipo* se almacenan todas las combinaciones posibles de pares de jugadores. De cada combinación se guarda el número de veces que han sido compañeros y el número de veces que han sido contrincantes. Cada combinación se identifica con el par de jugadores que la forman.

Los campos derivados de *ganados* y *perdidos* en la tabla de *Jugador*, permiten que durante la generación de la página de ranking solo con acceder a la tabla Jugador tengamos todo lo necesario. Esto mejora la consulta, que se va a realizar muchísimas veces, a costa de tener que almacenar estos campos y de tener que calcularlos cada vez que se introduce un nuevo partido, operación que se realizará el administrador muchas menos veces.

De la misma forma, los campos derivados de *compañeros* y *contrincantes* en la tabla *Equipo* nos evitan tener que recorrer todos los partidos cuando se genera la página que muestra la tabla de equipos.

En la tabla *Calendario* se almacenan todos los eventos que crea el administrador con un identificador interno, el tipo de evento, su fecha y el mensaje asociado.

Por último existe la tabla *Foro* donde quedan registrados todos los mensajes que se envían al servidor con un identificador interno, el *nick* y el mensaje introducido por el usuario, la fecha en que se envió, la ip del ordenador que lo envió y un booleano que indica si se mostrará el mensaje en la lista (pueden ocultarse mensajes que, por ejemplo, infrinjan las reglas del foro).

## 4.5 – Diseño de GesPartidos

El asistente de GesPartidos se ejecutará en una plataforma (J2ME) de recursos limitados. El diseño debe ser simple y por eso no tiene sentido dividirlo en niveles ni en capas. De hecho la plataforma no lo permite (no esta pensada para eso) ni tampoco es nuestra intención. Va a ser solo una aplicación (un *midlet* en la terminología de este ámbito) que constará de un conjunto de clases que se compilan todas al mismo tiempo, es decir, no se pueden crear librerías propias. Siempre es bueno separar las distintas partes de nuestra aplicación aunque no lleguemos a tener capas independientes e intercambiables. La única forma de hacer algo así es mediante clases.

Primero vamos a realizar el diseño de la interfaz de acuerdo al caso de uso que hemos trazado en la fase anterior de análisis, y luego veremos cuales son las clases que tendrá la aplicación, ya en la fase de diseño, especificando los principales métodos que deberán implementarse en cada una.

### 4.5.1 – Diseño de la interfaz

Al iniciar el asistente deberá leerse el fichero de entrada y, en un primer paso, se mostrará la lista completa de jugadores.

En esta pantalla, el administrador podrá marcar los jugadores que hayan asistido, tras lo cual puede avanzar al siguiente paso o salir de la aplicación sin más.


*Figura 22: Selección de jugadores*

Mientras se generan los partidos según la selección de jugadores que se haya realizado, aparecerá una barra de progreso. Más o menos como se muestra en la figura 23.

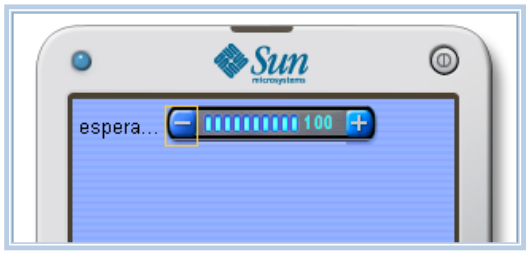


*Figura 23: Barra de progreso*

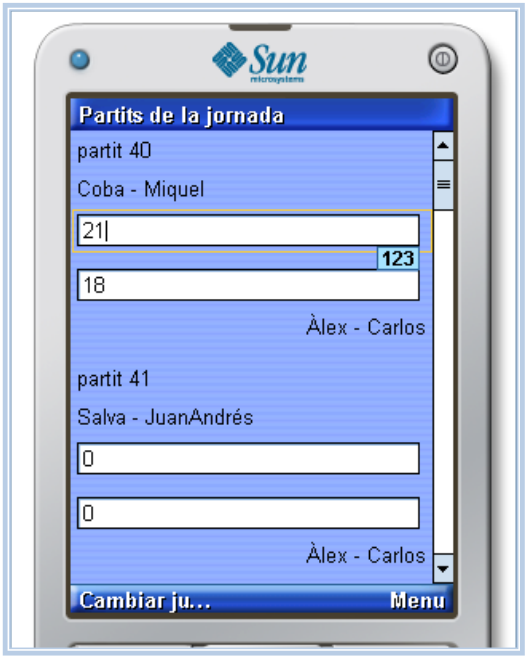


*Figura 24: Partidos generados*

Los partidos generados se muestran numerados y en el orden en que deben ser jugados para que todos los participantes intervengan por igual.

En el ejemplo de la figura 24, el primer partido de la jornada sería el número 40 de la liga y en el competirían el equipo formado por Coba y Miquel contra Alex y Carlos. En cuanto termina el partido se apuntan los marcadores de cada equipo en la casilla correspondiente y se pasa al partido siguiente.

En cualquier momento podemos volver a la pantalla de selección de jugadores para añadir un jugador que llegue más tarde o para quitar un jugador que se marche. Si hacemos esto, los partidos que ya se han jugado (y se han rellenado los marcadores) se conservarán en la lista y los restantes serán reemplazados por una nueva lista de quince nuevos partidos generados a partir del nuevo conjunto de jugadores y además teniendo en cuenta en la heurística los partidos ya jugados.

Al pulsar en el menú podremos acceder al resto de opciones que son: finalizar el asistente guardando los resultados de la jornada en un fichero, o cerrar el asistente sin guardar.

El fichero de salida se nombrará con la fecha actual y siguiendo la siguiente plantilla: resAAAAMMDDHHMM.sql (donde AAAA es el año, MM es el mes, DD es el día, HH es la hora y MM es el minuto).

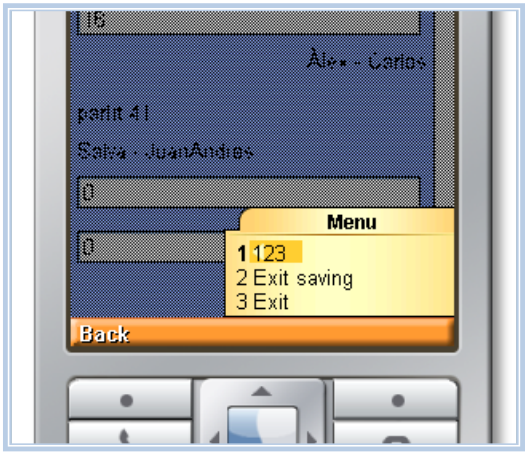


*Figura 25: Opciones ocultas del menú*

Un detalle a tener en cuenta es el siguiente, que se añadió al diseño tras detectar la deficiencia al realizar pruebas en diferentes móviles.

El fichero de entrada se busca por defecto en la siguiente ruta: *root1/other/backupLliga.xml* (donde *root1* es la primera unidad del sistema de ficheros que suele ser la de memoria interna del dispositivo). El administrador debería ubicarlo ahí si quiere automatizar el inicio del asistente. Si no lo hace o no puede hacerlo (hay dispositivos que restringen el acceso al sistema de ficheros y sobre todo a la unidad primaria), nada mas iniciar aparecerá un error como el de la Figura 27. Si se pulsa en "Buscar", se podrá explorar el sistema de ficheros para seleccionar el fichero de entrada que, de esta forma podrá estar en otro directorio y podrá tener cualquier nombre.

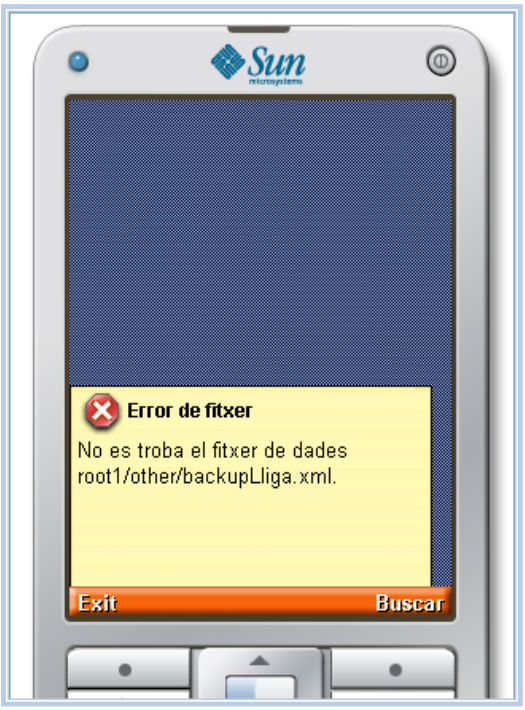


*Figura 27: Error de fichero de entrada*

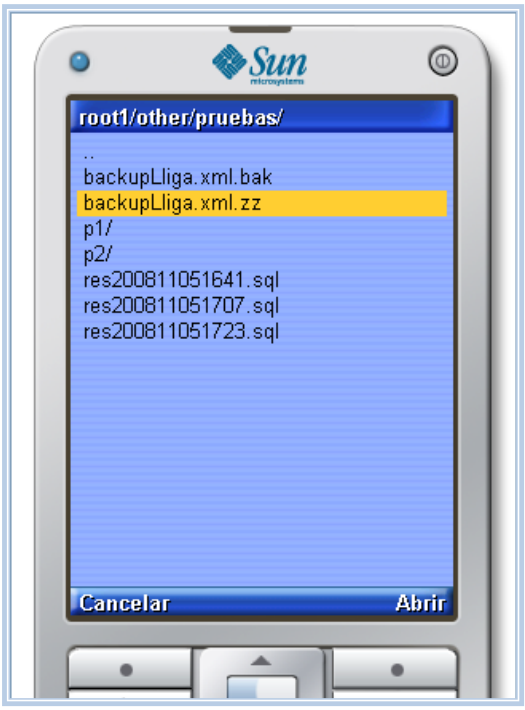


*Figura 26: Explorador de ficheros*

Una vez que se haya abierto el fichero de entrada, se mostrará la pantalla de selección de jugadores.

### 4.5.2 – Lógica de la aplicación

En la fase anterior se identificaron un conjunto de entidades que se debían incluir en el asistente a nivel conceptual. En esta fase de diseño vamos a ver que clases tendremos que implementar y que métodos deberán tener.

Usaremos de nuevo un diagrama de clases pero más detallado (en la figura 28) y además un diagrama de secuencia (en la figura 29).

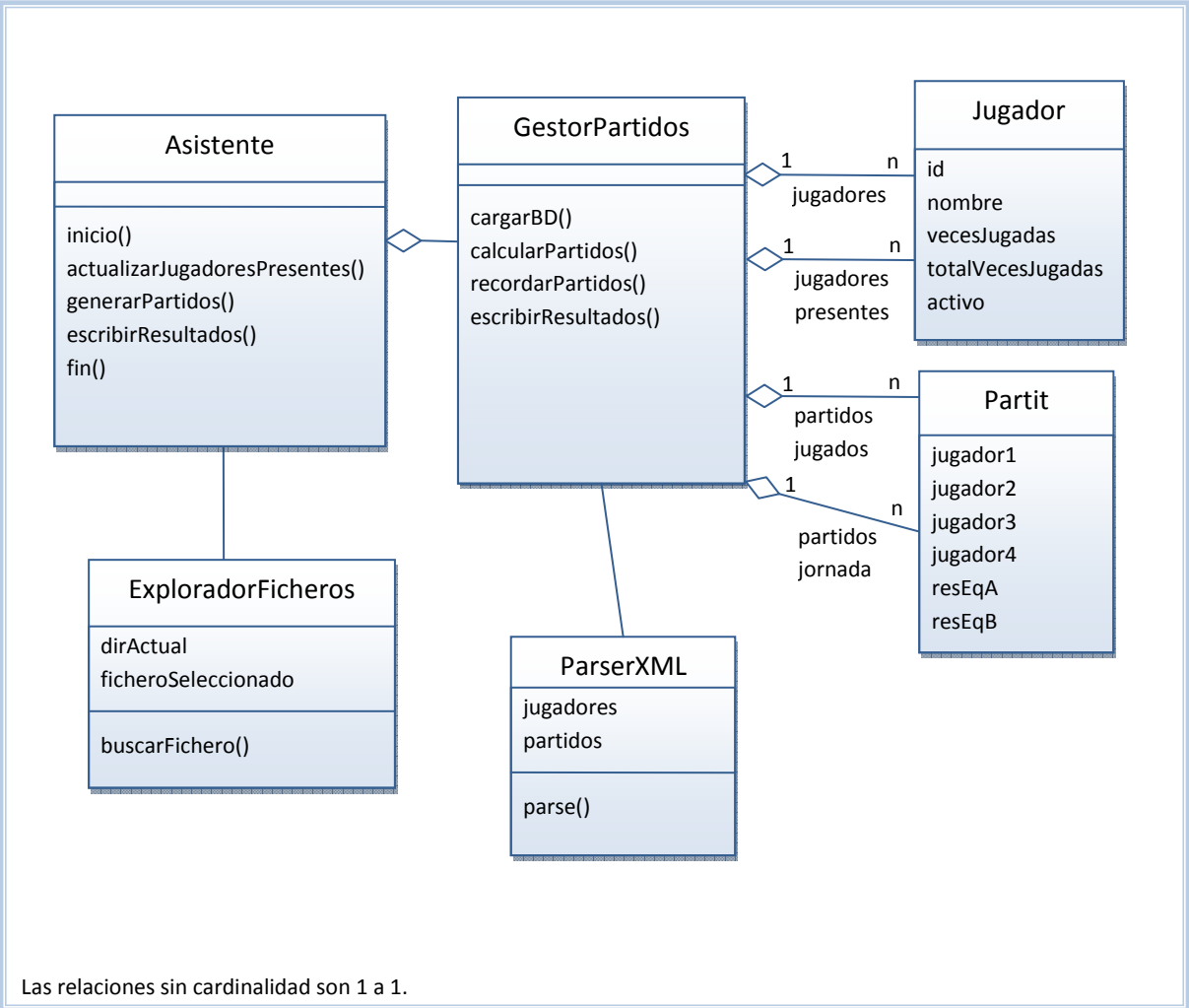


*Figura 28: Diagrama de clases*

El diagrama de secuencia complementa el diagrama de clases y se crea a partir de este junto con el caso de uso que vimos en la fase de análisis. En el diagrama de secuencia vemos el comportamiento dinámico del sistema: la interacción del usuario con la aplicación y de unas clases con otras a lo largo del tiempo de ejecución (el tiempo va de arriba a abajo).

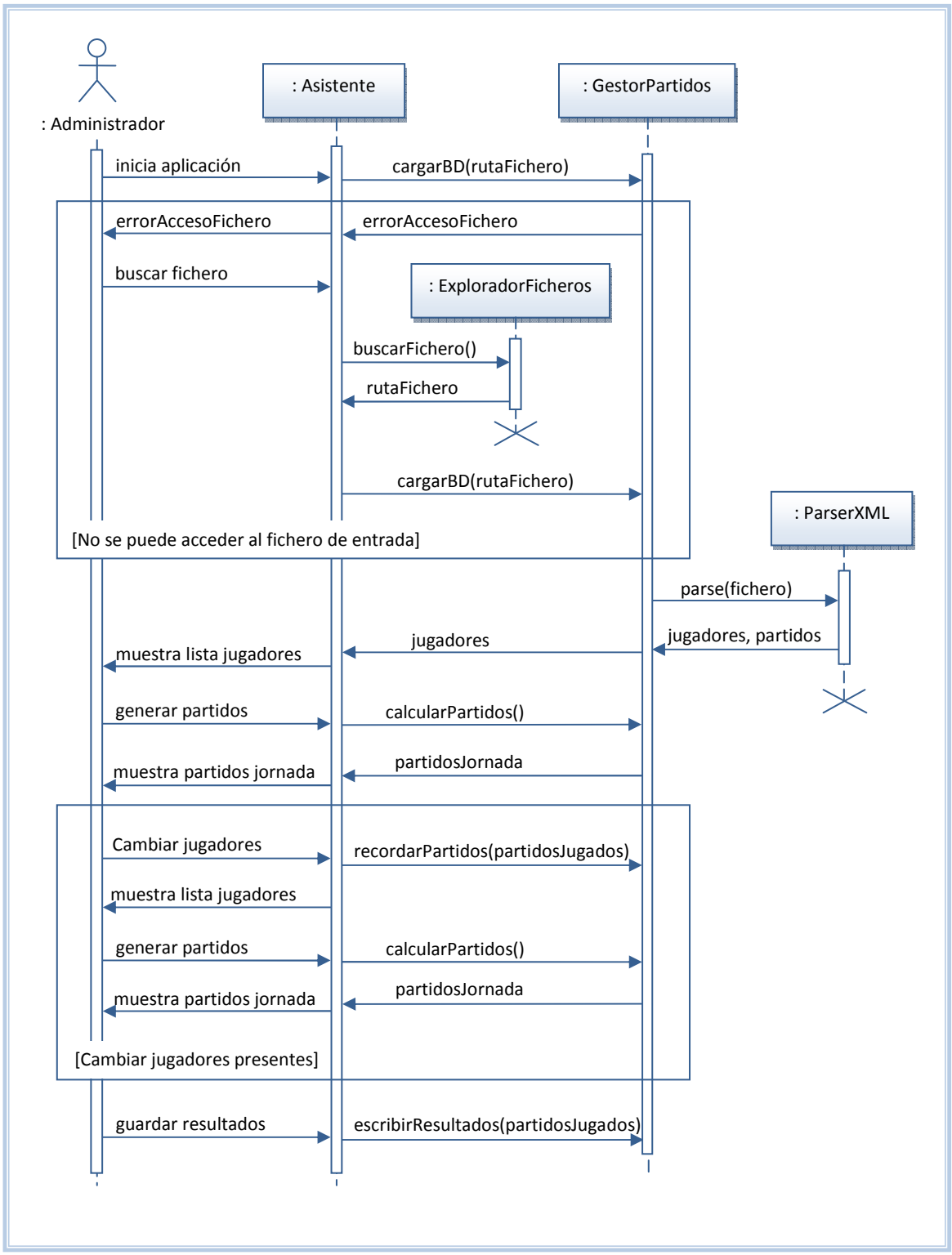


*Figura 29: Diagrama de secuencia*

Ya solo falta implementar las clases y sus métodos para que se comporten como en el diagrama y realicen cada una su función.

# 5 – Implementación

## 5.1 – Tecnologías

Algunas ya se han mencionado por ser un requisito de la aplicación. En el diagrama de la figura 30 se recogen todas las tecnologías y estándares empleados en la aplicación web, se clasifican en los distintos niveles y los interfaces de comunicación entre niveles.

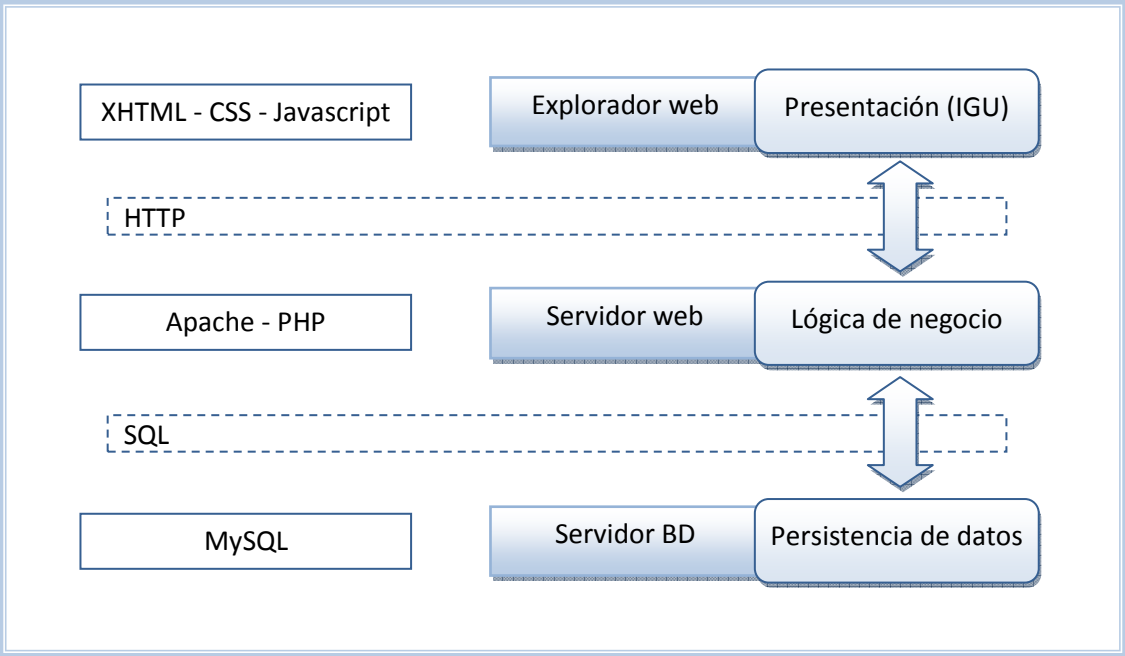


*Figura 30: Tecnologías empleadas en la parte web*

Para construir la interfaz gráfica de usuario, que se mostrará en el explorador web y que en definitiva son las distintas páginas web de que consta la aplicación, usaremos XHTML en combinación con CSS.

Todo explorador web conoce **XHTML** (*eXtensible HyperText Markup Language*), es un lenguaje de marcado que describe documentos en pantalla con el uso de etiquetas. Permite definir tanto la **estructura** como el **contenido** en texto, vínculos de hipertexto, imágenes, formularios y demás componentes de las páginas web.

XHTML está pensado para sustituir a HTML como estándar para las páginas web. En su versión 1.0, XHTML es solamente la versión XML de HTML, por lo que tiene, básicamente, las mismas funcionalidades, pero cumple las especificaciones, más estrictas, de XML. La adaptación a XML implica múltiples ventajas. Al ser más estricto permite simplificar el *parser*, y además podemos utilizar todas las herramientas creadas para procesamiento de documentos XML como editores, analizadores, buscadores, el lenguaje de transformación XSLT, validación con DTD, etc.

Con XHTML también podemos definir el **formato** sin embargo es mejor usar **CSS** (*Cascade Style Sheet*). Esta tecnología nos permite mayor expresividad y sobre todo separar el contenido de la forma. Con CSS definimos atributos (tamaño, color, borde, fuente, etc.) sobre las etiquetas de XHTML. Estos atributos además se heredan dentro de la jerarquía de etiquetas.

Las hojas de estilo se pueden insertar directamente en el mismo fichero XHTML o bien en un fichero de hoja de estilos, al que se enlazará en la cabecera. De esta segunda forma se centraliza toda la información de estilo en un único punto, con el que ganamos en eficiencia ya que los cambios se limitan a un único fichero y se reduce el número de líneas de los ficheros XHTML. Un cambio en la hoja de estilos afecta a todo el portal, mientras que de otra forma habría que revisar uno a uno todos los documentos.

Con CSS, una página puede incluso disponer de diferentes hojas de estilo según el dispositivo que la muestre o también a elección del usuario.

**Javascript** es un lenguaje que permite implementar las funciones de validación e interacción con el usuario y que ejecuta el navegador web. Como se ha comentado anteriormente, es una pequeña parte de la lógica de la aplicación que se traslada al nivel de presentación. Se trata de código que genera y envía el servidor web junto con la página web y que será interpretado y ejecutado por el navegador web en respuesta a determinadas acciones del usuario.

Para el servidor web, tanto el que se usará para las pruebas como el que nos proporciona la empresa de *hosting*, haremos uso de la implementación de **Apache**. Es lo que se viene usando en la mayoría de los servidores de Internet desde hace años en alguna de sus versiones.

En cada transacción que se produce entre un explorador web y el servidor se utiliza el protocolo **HTTP** (*Hypertext Transfer Protocol*). Este protocolo define la sintaxis y la semántica de los mensajes que se transmiten a través de la red. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. Las peticiones se efectúan sobre recursos que almacena el servidor y cada uno se identifica mediante un URL (localizador uniforme de recursos). Los recursos pueden ser de cualquier tipo y formato: texto, imagen, video,…; cualquier información e incluso puede se puede cifrar.

HTTP define 8 métodos de petición entre los que se encuentran los de GET y POST que se usan en este proyecto: GET para solicitar páginas pudiendo pasar algunos parámetros contenidos en la misma URL, y POST para cuando hay que transmitir al servidor gran cantidad de datos e incluso ficheros (como los que genera el asistente GesPartidos y que el administrador tiene que subir para actualizar la base de datos).

Para implementar la lógica de negocio de esta aplicación se usará **PHP**. Es un lenguaje de programación interpretado de propósito general aunque esta especialmente diseñado para desarrollo web. Se incrusta en el código XHTML de las páginas web de tal forma que el servidor web, cuando se le solicita una página y justo antes de enviarla, lee el fichero interpretando todo el código PHP que encuentra, ejecutándolo y volcando el resultado en la misma página que luego envía. Para conseguir esto hay que instalar un módulo en Apache.

El código fuente escrito en PHP es invisible al navegador y al cliente ya que es el servidor el que se encarga de ejecutar el código y enviar su resultado XHTML al navegador. Ningún código PHP es enviado al navegador. Esto hace que la programación en PHP sea segura y confiable.

Con PHP se puede emplear el paradigma de programación orientada a objetos. Uno de los puntos fuertes de este lenguaje es que permite la conexión a muchos tipos de servidores de bases de datos entre los que se encuentra MySQL, que es la que usaremos en este proyecto. Posee funciones para conectar, efectuar consultas sobre la base de datos y recuperar los resultados. PHP es libre, multiplataforma y posee una Biblioteca nativa de funciones sumamente amplia.

Como ya hemos avanzado, usaremos el gestor de bases de datos relacional **MySQL**, pues es lo que nos ofrece la empresa de *hosting* que hemos contratado y de hecho es de los que más se usan para portales web ya que es sencillo y muy rápido en lectura pero malo en entornos de alta concurrencia en la modificación (y la web no es uno de estos entornos). Es un servidor multihilo, multiusuario y es software libre.

Para formular las consultas a la base de datos desde el código PHP usaremos el lenguaje **SQL**, estandarizado ISO con algunas pocas variaciones propias de MySQL. Es un lenguaje declarativo de alto nivel. En la actualidad el SQL es el estándar de facto de la inmensa mayoría de los SGBD comerciales.

### 5.1.1 – Tecnologías empleadas en GesPartidos

El asistente GesPartidos se ejecuta en la plataforma J2ME (o Java ME). Esta tecnología proporciona un entorno robusto, flexible y portable para ejecutar aplicaciones sobre teléfonos móviles, PDAs o electrodomésticos; dispositivos con recursos hardware bastante inferiores a los de un PC.

Los dispositivos que con J2ME incorporan, según el ámbito al que pertenecen, una configuración, un perfil y algunos paquetes opcionales.

La configuración se compone de una máquina virtual y un conjunto mínimo de librerías con sus APIs (Application Program Interface) que proporcionan un nivel mínimo de funcionalidad para poder desarrollar aplicaciones. Existen dos configuraciones actualmente:

- CLDC (Connected Limited Device Configuration) para dispositivos con conexiones de red intermitentes, procesadores lentos y memoria limitada: teléfonos móviles, PDAs, etc. Incluye una máquina virtual muy reducida denominada KVM (Kilo Virtual Machine, debido a que requiere sólo unos pocos Kilobytes de memoria para funcionar).
- CDC para dispositivos con más memoria, procesador más rápido y un ancho de banda mayor. Incluye una máquina virtual Java completa (Java Virtual Machine, JVM) y un subconjunto de APIs de la arquitectura J2SE mucho mayor.

Los perfiles son librerías de más alto nivel que controlan el ciclo de vida de la aplicación, la interfaz de usuario y el acceso a propiedades especificas de los dispositivos. Los perfiles identifican un grupo de dispositivos por la funcionalidad que proporcionan y el tipo de

aplicaciones que se ejecutarán en ellos. Para CLDC existen dos perfiles: PDA Profile y Mobile Information Device Profile (MIDP).

La mayoría de teléfonos móviles incorporan J2ME con la configuración CLDC 1.1, el perfil MIDP 2.0 y algunos paquetes opcionales que amplían la funcionalidad. En esta aplicación se usa dos: el paquete JSR-75 (FileConnection) que proporciona acceso al sistema de ficheros en el dispositivo y tarjetas de memoria, y el JSR-172 dedicado a los servicios web.

Las aplicaciones J2ME desarrolladas bajo la especificación MIDP, se denominan *midlets*. Las clases de un *midlet* se compilan produciendo ficheros *.class* que se empaquetan en un fichero *.jar*, junto con otros recursos (imágenes, sonidos, …) y un fichero de *manifiesto* que describe la aplicación.

El entorno de ejecución de los *midlets* es diferente al de una aplicación java normal. La función principal del *midlet* no es la función *main()* y tampoco está permitido que sea el *midlet* el que termine la ejecución de la máquina virtual. La clase principal debe heredar de *javax.microedition.midlet.MIDlet* e implementar los métodos abstractos startApp(), pauseApp() y destroyApp(). Estos métodos permiten al *midlet* cambiar de estado. El gestor de aplicaciones (AMS) cambia su estado haciendo una llamada a cualquiera de los métodos anteriores y el mismo *midlet* también puede cambiar su propio estado.

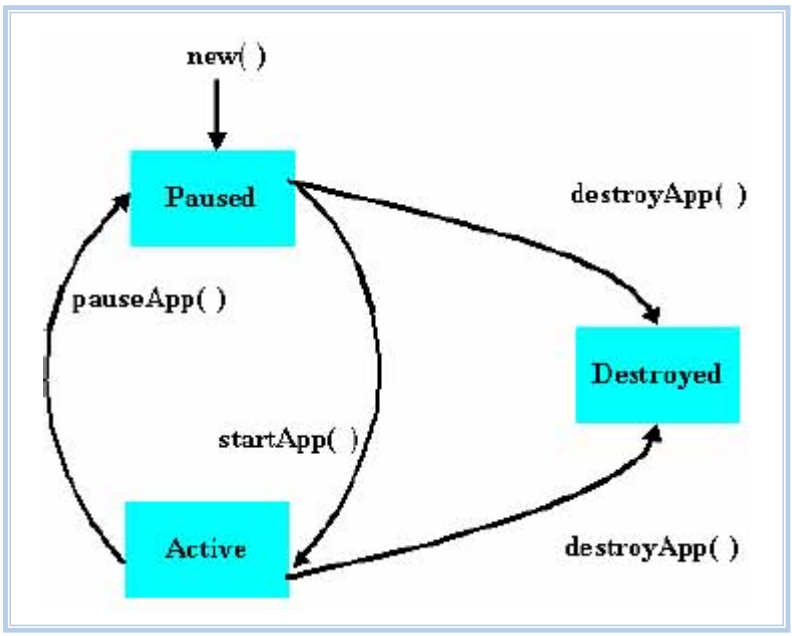


*Figura 31: Diagrama de estados de un midlet*

## 5.2 – Herramientas

Para el desarrollo e implementación de esta aplicación se han empleado diversas herramientas que vamos a comentar a continuación. Todas son gratuitas y pueden descargarse fácilmente desde la web del autor.

Para editar el código de las página web tanto, en XHTML como en PHP o javascript, sirve cualquier editor de textos aunque si tiene resaltado de sintaxis mucho mejor, como el **Notepad++** (v5.5.1) ( *http://notepad-plus.sourceforge.net/es/site.htm*). Hay otras herramientas más avanzadas como Dreamweaver que ofrecen mayor productividad, tienen autocompletado, permiten diseñar la web en modo gráfico escribiéndose el código automáticamente y hasta permiten ver el aspecto más o menos final de la web. Sin embargo uno de los principales objetivos del proyecto era aprender el funcionamiento de las tecnologías web experimentando con los distintos lenguajes de programación y por eso hemos optado por la primera opción.

Mediante el paquete **Xampp** (v1.6.7) (http://www.apachefriends.org/es/xampp.html) podemos instalar, en el mismo ordenador con el que desarrollamos, un servidor web **Apache** (con el módulo que interpreta PHP, el conector a bases de datos de MySQL y el módulo SSL) y un servidor **MySQL**, todo a la vez en el mismo instalador. Es multiplataforma. Con esto y un explorador podremos visualizar al instante el resultado de nuestro desarrollo sin tener que subirlo a un servidor en Internet.

Xampp incorpora también, entre otras cosas, **phpMyAdmin** que es un cliente web de bases de datos MySQL que nos servirá para gestionar el contenido de la base de datos de nuestra aplicación durante el desarrollo y el mantenimiento.

Para el diseño visual de la interfaz web resulta muy útil un complemento (o *plugin*) que existe para el navegador **Mozilla Firefox** llamado **Firebug** (v1.5.3). Con él podemos ver cómo es el código XHTML y CSS que llega realmente al explorador, localizando fácilmente cada componente en el código, e incluso depurarlo en el mismo navegador, pues permite modificar el código actualizando la visualización de la web conforme lo cambiamos.

En la implementación de la aplicación para móviles GesPartidos se usó **NetBeans IDE 6.8** (http://netbeans.org/). Es un completo entorno de desarrollo originalmente para Java, pero que ahora abarca otros muchos lenguajes y plataformas, entre ellos J2ME para desarrollar midlets como el nuestro. Nos permite escribir, compilar y hasta ejecutar la aplicación sobre un dispositivo virtual testeándola así sin tener que instalarla cada vez en el dispositivo móvil físico.

## 5.3 – Detalles de implementación

En este apartado vamos a conocer los detalles más relevantes de la implementación realizada. Toda la implementación (ficheros de código, recursos, etc.), tanto de la parte web como del asistente GesPartidos, se adjunta a esta memoria. Aquí solo se citarán ciertas partes de código para visualizar lo que se explica.

### Interfaz

Empecemos por la interfaz de usuario. Como ya sabemos está hecha en XHTML y CSS. Estas dos tecnologías se "solapan", de tal forma que es posible implementar completamente una web sin CSS, es decir, XHTML incluye a CSS; pero no lo incluye completamente ya que este último permite hacer cosas que XHTML no puede.

Una web en general tiene tres partes: estructura, contenido y estilo (o apariencia). Para definir el contenido (en este proyecto) se usa XHTML, para el estilo lo mejor es CSS, y para la estructura o "maquetación" se usan ambos a la vez. En esto hay diversidad de opiniones, últimamente se aconseja usar CSS siempre que sea posible. En este proyecto se ha intentado seguir esta recomendación.

Como hemos visto ya en la fase anterior, la estructura es la misma en todas las páginas. Se usa siempre el mismo código que se encuentra en *inc/head.php* y en *inc/foot.php*. Estos dos ficheros se importan siempre al principio y al final (respectivamente) en todas las páginas.

El código es el siguiente:

inc/head.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="es-CA" lang="es-CA" xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<link rel="stylesheet" type="text/css" href="<?php echo ROOT ?>webdesign.css?id=24" />
<title>booola!!</title>
</head>
<body>
<div class="wrapper">
  <div class="footer">
    <div class="header">
      <div class="content">
<div id="title">UPV Frontenis Champinyon's League</div>
<div id="left">
  <div class="menu_item">
    <a href="<?php echo ROOT ?>publicPages/ranking.php" class="menu_link">Ranking</a></div>
  <div class="menu_item">
    <a href="<?php echo ROOT ?>publicPages/calendari.php" class="menu_link">Calendari</a></div>
  <div class="menu_item">
    <a href="<?php echo ROOT ?>publicPages/foro.php" class="menu_link">miniForum</a></div>
  <div class="menu_item">
    <a href="<?php echo ROOT ?>publicPages/normativa.php" class="menu_link">Normativa</a></div>
```

```
    <div class="menu_item">
      <a href="<?php echo ROOT ?>publicPages/equips.php" class="menu_link">Equips</a></div>
    <div class="menu_item">
      <a href="<?php echo ROOT ?>publicPages/estadistiques.php" class="menu_link">Estadístiques</a></div>
    <div class="menu_item"></div>
  </div>
```

inc/foot.php

```
        </div>
      </div>
      <div class="webdesign">Web design by <a href="mailto:palollo@hotmail.com" title="Web
design">palollo</a></div>
    </div>
  </div>
  </body>
  </html>
```

En la figura 32 podemos ver de forma visual la distribución de los * * que se ven en el código.


*Figura 32: Maquetación de la web*

Observando el código anterior puede verse que, siguiendo la recomendación, en ninguna etiqueta se incluye el atributo *style*, y tampoco se utilizan tablas para maquetar (colocar cada parte en su sitio). Todo esto se hace con CSS estableciendo las propiedades de cada etiqueta según su clase en el fichero *webdesign.css*. Dicho fichero aglutina todos los estilos del portal (excepto los del foro, que va aparte como se explicará después). Seguidamente se muestran solo los estilos que afectan a la maquetación de la estructura general de las páginas.

webdesign.css

```
[…]

.wrapper {
        width: 966px;
        background-image: url(images/middle.jpg);
        background-repeat: repeat-y;
        margin: auto;}
.header {
        width: 966px;
        background-image: url(images/header.jpg);
        background-repeat: no-repeat;
        margin-bottom: 20px;
        padding-bottom: 30px;}
.footer {
        width: 966px;
        background-image: url(images/footer.jpg);
        background-repeat: no-repeat;
        background-position: bottom;}

.content {
        width: 966px;
        overflow: hidden;}
#title {
        margin: 50px 102px 15px 102px;
        color: #60A460;
        font-weight: bold;
        font-size: 140%;
        text-align: left;
        text-transform: capitalize;}
#left {
        float: left;
        width: 154px;
        padding: 10px 27px 10px 102px;
        text-align: left;}
#right {
        float: left;
        width: 582px;
        margin: 350px 0px 0px 0px;
        padding: 0px;
        text-align: justify;
        min-height: 168px;
        overflow: auto;}
```

```
.menu_item {
        background-image: url(images/menu_bg.gif);
        background-repeat: repeat-x;
        padding: 9px 2px 8px 2px;}
.webdesign {
        color: #588EC8;
        background-color: inherit;
        text-align: right;
        margin-right: 62px;}

[…]
```

Cada *div* lleva una imagen de fondo que hay que editar con alguna herramienta (se usó Gimp, que es gratuito y muy completo) para que salgan bien cuadradas y no se noten los bordes. Un detalle delicado en esta tarea es conseguir que se pueda alargar la página verticalmente sin que se deforme la estructura ni la imagen de fondo. Esto se consigue con el *div* llamado *wrapper* que está centrado en el fondo, solo se ve cuando no hay ninguna capa por encima, que es cuando termina el *header* y antes de que empieze el *footer*. El *wrapper* lleva asociado una imagen de fondo que se repite verticalmente de forma indefinida y, para que no se noten los trozos, se igualan los bordes superior e inferior de dicha imagen.


*Figura 33: Partes de la imagen de fondo*

### Lógica de negocio

A cada página se le asocia un fichero *.php* que la genera. A continuación podemos ver el código de generación de la página de *ranking*.

publicPages/ranking.php

```
<?php
include('path.php');
include(ROOT.'inc/head.php');
include(ROOT.'inc/bd.php');
?>

<div id="right">
<h1>Ranking</h1>
<br />

<?php
    $db = conectarBD() or die;
    $query = "SELECT * FROM jugador ORDER BY guanyats DESC, perduts ASC";
    $result = mysql_query($query, $db);
    if($result == FALSE)
      $num_results = 0;
    else
      $num_results = mysql_num_rows($result);

    echo "<table id=\"ranking\" cellspacing=\"0px\" >
            <tr> <th> </th> <th>punts</th> <th>guanyats</th> <th>perduts</th>
                 <th>jugats</th> <th>pilotades rebudes</th> </tr>\n";

    for($i=0; $i<$num_results; $i++){
        $row = mysql_fetch_array($result);
        echo "<tr "; if(!$row['valid']) { echo "class=\"ratllat2\" "; $desactivats=true; } ; echo ">";
        echo "     <th>" . $row["nom"] . "</th>
                   <td>" . ( $row["guanyats"] * 2 ) . "</td>
                   <td>" . $row["guanyats"] . "</td>
                   <td>" . $row["perduts"] . "</td>
                   <td>" . ( $row["guanyats"] + $row["perduts"] ) . "</td>
                   <td>" . $row["pilotades"] . "</td>
                </tr>";
    }

    echo "</table>";
    if($result != FALSE) mysql_free_result($result);
    mysql_close($db);
    if($desactivats) echo "<br/>Els jugadors ratllats són jugadors desactivats per falta d'assitència continuada. ".
                          "Més informació a la secció de <a href=\"normativa.php\">Normativa</a>.<br/>";
?>

</div>
<?php include(ROOT.'inc/foot.php') ?>
```

El resto de páginas sigue más o menos el mismo patrón tal como se explica en el capítulo anterior (fase de diseño).

El fichero *path.php* se encuentra siempre en todos los directorios y se incluye siempre en el código. Tan solo define la variable *ROOT* que almacena la ruta al directorio raíz de la aplicación y permite hacer referencia a los recursos (como una imagen u otro fichero de código) usando una ruta absoluta. Si se usa la variable *ROOT,* en cualquier momento podemos cambiar de sitio los ficheros de código de las páginas sin tener que ir buscando y cambiando todas las referencias, solo habría que generar un fichero *path.php* en el nuevo directorio.

En el directorio *inc* se encuentran todos los ficheros con código reutilizable y que, por tanto, se incluyen en muchas de las páginas. Se ha hablado ya de *head.php* y *foot.php*, y el siguiente que hay que mencionar es *bd.php*, que contiene funciones para conectar o consultar a la base de datos. La función que conecta con la base de datos se utiliza en la página de *ranking*, cuyo código acabamos de ver, y en todas aquellas en las que se muestre información almacenada en la base de datos.

inc/bd.php

```
function conectarBD() {
    global $servidorBD, $userBD, $passwordBD, $nomBD;

        @ $db = mysql_pconnect($servidorBD, $userBD, $passwordBD);
        if ($db) {
                @ mysql_select_db($nomBD, $db);
                return $db;
        } else {
                echo "<br/>Error: no es pot connectar amb la base de dades. Intente fer-ho més tard.<br/>";
                return false;
        }
}
```

Otro de los puntos interesantes a comentar es que toda la configuración se aglutina en un fichero que se llama *configuracion.php*. De esta forma, para actualizar el software se sustituyen todos los ficheros excepto este. De momento solo se definen las cuatro variables que son las que se usan en la función que acabamos de ver para conectar con la base de datos.

El foro se ha implementado como un 'módulo' reutilizable en otras aplicaciones. Todos los ficheros que intervienen en la generación del foro se encuentran en el directorio *publicPages/foro*. El foro tiene sus propios recursos, su propio fichero de estilos CSS y su propio fichero de configuración. La conexión se encuentra en publicPages/foro.php que se muestra a continuación:

publicPages/foro.php

```
<?php
include('path.php');
```

```
include(ROOT.'inc/head.php');
?>

<div id="right">
    <h1>miniForum</h1>
    <br />
    <iframe src="foro/chatforo.php" width="100%" height="582" frameborder="0" scrolling="no" ></iframe>
</div>

<?php include(ROOT.'inc/foot.php') ?>
```

Como vemos, se incrusta la página generada por el 'módulo' como un *iframe* en la zona de contenido de la página de foro de GesLigas. Al hacerlo de esta forma, cuando el usuario pulse en el botón de refrescar del foro para comprobar si hay nuevos mensajes, solo se recarga el *frame* y no toda la página. Por otra parte, el componente, se ha diseñado de tal forma que se acopla al tamaño del *frame* en el que se encuentra.

Otro detalle interesante que cabe mencionar es la posibilidad de incluir código PHP en el contenido de los eventos del calendario tal como vemos en la figura 34.

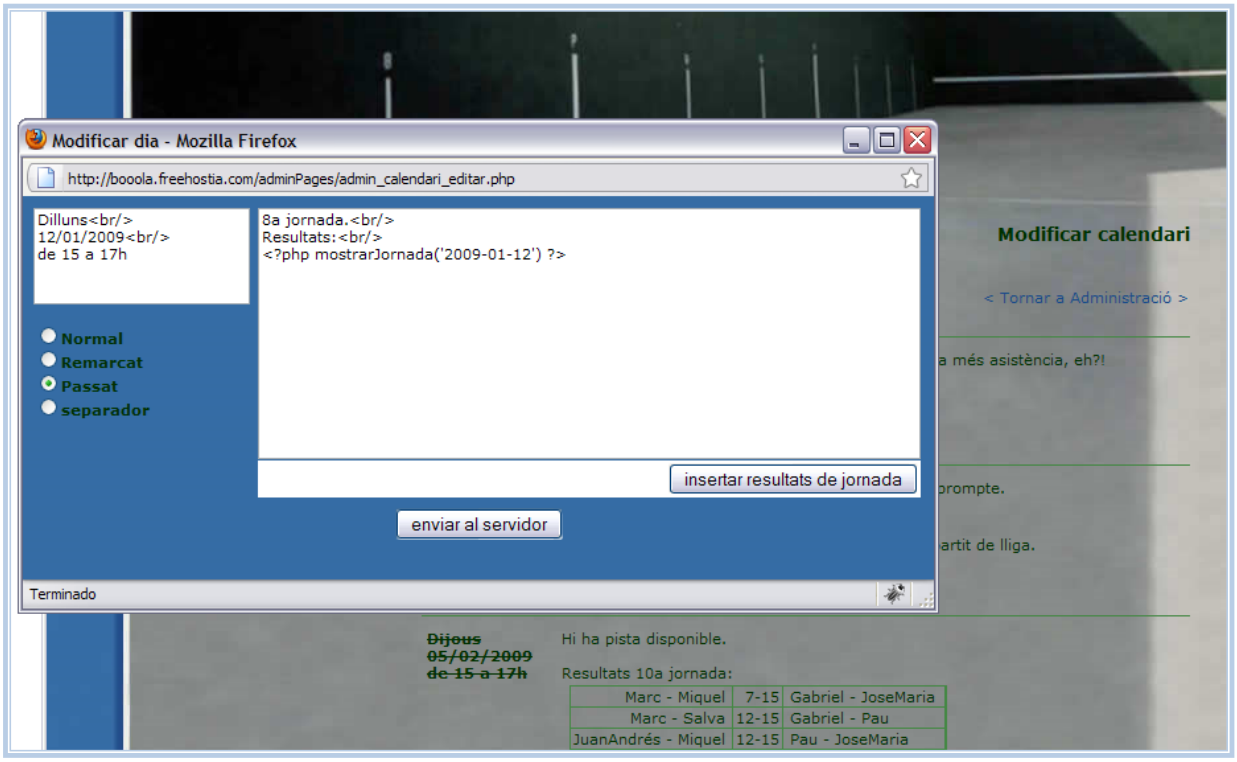


*Figura 34: Ventana de modificación de un evento*

De esta forma el usuario administrador puede insertar ciertos 'componentes' en el mensaje, como por ejemplo una tabla con los resultados de una jornada determinada. Al insertar el 'componente' (pulsando en el botón correspondiente) realmente lo que se inserta es una llamada a una función, ya definida en la aplicación, que se evalúa cuando se muestra el calendario. En el caso de la tabla de resultados de jornada (que es el único componente que

hay por el momento) se accede a la base de datos para recuperar la información y se genera la tabla en XHTML.

Para conseguir esto se usa el siguiente código:

publicPages/calendari.php

```
[…]

<?php
function mostrarJornada($data){
    global $db, $jugador;
    if (!$db) return false;
    $query = "SELECT * FROM partit WHERE dia='" . $data . "' ORDER BY num ASC";
    $result = mysql_query($query, $db);
    if($result == FALSE) return false;
    $num_results = mysql_num_rows($result);
    if ($num_results > 0) {
        echo "<table class=\"resultats\" cellspacing=\"0px\">\n";
        for($i=0; $i<$num_results; $i++){
            $row = mysql_fetch_array($result);
            echo "<tr";
            if (!$row["valid"]) echo " class=\"ratllat2\" ";
            echo "><td class=\"esq\">" .
                    $jugador[$row[jug1]] . " - " . $jugador[$row[jug2]] .
                    "</td> <td>" .
                    $row[resultatEqA] . "-" . $row[resultatEqB] .
                    "</td> <td class=\"dreta\">" .
                    $jugador[$row[jug3]] . " - " . $jugador[$row[jug4]] .
                    "</td></tr>\n";
        }
        echo "</table>\n";
    }
    mysql_free_result($result);
}

function eval_buffer($string) {
    ob_start();
    eval("$string[2];");
    $ret = ob_get_contents();
    ob_end_clean();
    return $ret;
}

function processar($string) {    // evalua todo el codigo PHP que se encuentre en el string que se le pasa
    return preg_replace_callback("/(<\?php|<\?)(.*?)\?>/si", "eval_buffer", $string);
}
?>

[…]
```

La primera función que vemos es la que viene predefinida y se llama para mostrar la tabla de resultados de la correspondiente jornada. Para evaluar el código PHP que se encuentra en el mensaje de los eventos y así poder efectuar realmente la llamada a la primera función, se usan las otras dos que vienen a continuación. Con cada evento se llama a la función *processar()* pasándole como *string* el mensaje del evento. Dicha función localiza cualquier subcadena que se encuentre entre etiquetas que delimitan código PHP y se las pasa a la función *eval_buffer()*. Esta última utiliza la función *eval()* (que proporciona PHP) para evaluar el código y, con la ayuda de las funciones de "output buffering", devuelve el resultado de nuevo en forma de cadena que se insertará en el código XHTML final.

Queda pendiente ver como se implementa alguna de las funciones de validación de campos con Javascript en la parte cliente (lo ejecuta el explorador).

adminPages/index.php

```
[…]

<script language="javascript">

function esDigito(sc) {
    var sCod = sc.charCodeAt(0);
    return (sCod > 47) && (sCod < 58);
}

function validaFecha(f) {
    var ok = true;
    if (f.value != "") {
        ok = ok && esDigito(f.value.substr(0,4));
        ok = ok && esDigito(f.value.substr(5,2));
        ok = ok && esDigito(f.value.substr(8,2));
        if (!ok){
            alert("Fecha inválida.");
            f.focus();
        }
    }
}

[…]
</script>

[…]
<h2>Introduir un partit</h2>
<form name="formulario" method="post" action="mostrarConsulta.php?op=nouPartit" target="_self"
onsubmit="return validarPartit(this)">
<table>
    <tr>
        <td>data (YYYY-MM-DD)</td>
        <td> <input type=text name="dia" maxlength="10" size="11" onblur="validaFecha(this)"> </td>
    </tr>
[…]
```

En el código anterior vemos la validación de la fecha en la introducción manual de un partido (en la página de administración general). Podemos declarar funciones en lenguaje Javascript en cualquier parte dentro de la página XHTML pero siempre entre la etiqueta de comienzo *<script language="javascript">* y la de fin *</script>*. Luego para llamar a las funciones en respuesta a acciones del usuario se mete la llamada en el atributo correspondiente al evento al que queramos responder del componente que se valida. En este caso, se responde al evento *onblur* del campo de texto de la fecha (con nombre "dia") que se dispara cuando el foco abandona el campo.

### Persistencia

Para que la conexión, las consultas y las modificaciones que se realizan sobre la base de datos desde el servidor web mediante el código PHP, tengan éxito hay que crear primero la base de datos con sus tablas. Para hacer esto se puede usar cualquier cliente de MySQL como phpMyAdmin. Las operaciones que hay que realizar para crear las distintas tablas a las que accede la aplicación, se incluyen en el primero de los anexos de esta memória.

## 5.3.1 – Implementación de GesPartidos

En este apartado se van a explicar algunos detalles que hay que tener en cuenta a la hora de implementar un *midlet* en general y nuestro asistente de partidos en particular.

Cuando se inicia la aplicación, la máquina virtual crea una instancia de la clase principal, la que extiende de *MIDlet* e implementa la interfaz *CommandListener*, y que en este caso se llama *Assistent*. Este objeto se encargará de crear las distintas pantallas de la interfaz de usuario y atender a la acciones del usuario, delegando las tareas gestión de partidos en la clase *GestorPartits*.

En el creador de *Assistent*, que vemos a continuación, se guarda el *display* (al que enviaremos cada una de las pantallas que creemos y queramos mostrar), creamos nuestra instancia de *GestorPartits* (el objeto que realiza casi todo el trabajo) y llamamos al método privado *inici()*, que vemos más abajo.

Assistent.java

```
[…]

public class Assistent extends MIDlet implements CommandListener {
[…]

  public Assistent() {
      disp = Display.getDisplay(this);
      GP = new GestorPartits();
      inici();
  }

[…]
```

```
/* Es llig el fitxer de la BD.
 * Es crea i es mostra el formuari de selecció de jugadors. */
private void inici() {
    /* Llegir el fitxer XML que conté la base de dades (jugadors i partits) */
    try {
        GP.carregarBD(home + fitxerBD);
    } catch (Exception e) {
        errorFitxer(e.getMessage());
        e.printStackTrace();
        return;
    }

    /* crear el formulari de selecció de jugadors */
    formulariSeleccio = new Form("Selecció de jugadors");
    formulariSeleccio.append("Indica la gent que hi ha:");
    ChoiceGroup grupSeleccio =
        new ChoiceGroup("jugadors", Choice.MULTIPLE);
    Enumeration jugadors = GP.getEnumJugadors();
    while(jugadors.hasMoreElements()){
        Jugador j = (Jugador) jugadors.nextElement();
        grupSeleccio.append(j.getNom(), null);
    }
    formulariSeleccio.append( grupSeleccio );

    formulariSeleccio.addCommand( new Command("Exit", Command.EXIT, 0) );
    formulariSeleccio.addCommand( new Command("Seguir", Command.OK, 0) );
    formulariSeleccio.setCommandListener(this);

    disp.setCurrent(formulariSeleccio);
}

[…]
```

El método *inici()* da comienzo al asistente: manda al gestor de partidos que cargue el fichero de entrada (más adelante se explica cómo se lee el fichero XML) y construye la pantalla de inicio de selección de jugadores para luego mostrarla pasándosela al *display*.

Una "pantalla" es un objeto que hereda de *displayable*. Hay solo unos pocos y entre ellos está el formulario que es el que usamos en el código anterior para la pantalla de selección de jugadores. A cualquier *displayable* le podemos añadir comandos que se mostrarán en pantalla y que podrá accionar el usuario. Tras añadir los comandos de "Exit" y "Seguir" hay que establecer la clase *CommandListener* que atenderá a las acciones del usuario y en este caso es esta misma clase principal.

Para cumplir con la interfaz *CommandListener* hay que implementar el método *commandAction()* que decide qué se hace con cada comando que elige el usuario.

Assistent.java

```
[…]

public class Assistent extends MIDlet implements CommandListener {
[…]

  public void commandAction(Command c, Displayable s) {

    if ("Selecció de jugadors".equals(s.getTitle()) ) {
      if (Command.EXIT == c.getCommandType() ){   //Exit
        fi();
      } else
      if (Command.OK == c.getCommandType()){   //Seguir
        generarPartits();
      }
    } else
    if ("Partits de la jornada".equals(s.getTitle()) ) {
      if (Command.EXIT == c.getCommandType() ){   //Exit
        fi();
      } else
      if (Command.OK == c.getCommandType()){   //Exit saving
        escriureFitxerResultats();
      } else
      if (Command.BACK == c.getCommandType()){   //Cambiar jugadors
        cambiarJugadors();
      }
    } else
    if ("Error de fitxer".equals(s.getTitle())){
      if (Command.EXIT == c.getCommandType()) {   //Exit
        fi();
      } else
      if (Command.SCREEN == c.getCommandType()) {
        if ("Buscar".equals(c.getLabel())){   //Buscar
          buscarFitxer();
        } else
        if ("fiExploracio".equals(c.getLabel())) {   //fiExploracio
          //Este comando es executat per l'explorador de fitxer quan acaba.
          fitxerSeleccionat();
        }
      }
    } else
    if ("Error fatal".equals(s.getTitle())){
      if (Command.OK == c.getCommandType()) {
        fi();
      }
    }
  }

[…]
```

Según la pantalla en la que se encuentra el usuario y el comando accionado se llama a un método privado que generalmente manda al gestor de partidos las tareas que debe realizar y construye la pantalla donde se muestran los resultados.

Al cargar el fichero de entrada puede ocurrir que no se encuentre dicho fichero en la ubicación predeterminada, en cuyo caso se muestra un error en pantalla. En esta pantalla de error, el usuario podrá salir de la aplicación o buscar el fichero. Para incluir la segunda posibilidad se implementó un pequeño explorador del sistema de ficheros que permite ir entrando en los directorios, listando sus entradas y seleccionar finalmente el fichero que se busca. Se encuentra en la clase *ExploradorFitxers* y funciona del siguiente modo:

- Se crea una instancia pasándole al creador el *display* que usará para mostrar sus pantallas.
- Se llama al método *buscarFitxer()* pasándole un *commandListener* que atenderá al comando de fin de la exploración.
- Se recupera la ruta y el nombre del fichero seleccionado con el método *getFitxerSeleccionat()*.

Antes hemos visto como, para cargar el fichero de entrada, la clase principal llama al método *carregarBD()* de la clase *GestorPartits* que se muestra a continuación.

GestorPartits.java

```
[…]

public class GestorPartits {
[…]

  /* Ompli els jugadors i els partitsJugats a partir del fitxer d'entrada que
   * conté la base de dades en format XML.
   * Si no es pot llegir el fitxer es llança una excepció. */
  public void carregarBD(String nomFitxer) throws Exception {
      InputStream fis = null;
      FileConnection fc = null;
      try {
          fc = (FileConnection) Connector.open("file://localhost/" + nomFitxer, Connector.READ );
      } catch (Exception e) {
          throw new Exception(
              "No es pot accedir al fitxer de dades" + nomFitxer + ".\nException: " + e.getMessage() );
      }
      if ( !fc.exists() ) {
          throw new Exception("No es troba el fitxer de dades " + nomFitxer + "." );
      }
      try {
          fis = fc.openInputStream();
      } catch (Exception e) {
          throw new Exception(
              "No es pot accedir al fitxer de dades" + nomFitxer + ".\nException: " + e.getMessage() );
      }
```

```
        try {
            Parsejador parsejador = new Parsejador();
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser saxParser = factory.newSAXParser();
            parsejador.reset(jugadors, partitsJugats);
            saxParser.parse(fis, parsejador);
            numUltimPartit = parsejador.getNumUltimPartit();
        } catch (Exception e) {
            try {
                fis.close(); fc.close();
            } catch (Exception ex) {ex.printStackTrace();}
            throw new Exception(
                "No es pot interpretar el fitxer de dades" + nomFitxer + ".\nException: " + e.getMessage() );
        }
        try {
            fis.close(); fc.close();
        } catch (Exception e) {e.printStackTrace();}

[…]
  }

[…]
```

El fichero de entrada contiene la base de datos en formato XML. Para leerlo se usa un objeto *SAXParser*, proporcionado por el paquete opcional JSR-172. El objeto *SAXParser* parsea un *InputStream* abierto sobre el fichero de entrada, llamando con cada elemento que encuentra a los métodos que implementa la clase *Parsejador* impuestos por la interfaz *DefaultHandler*:

- *startElement()* al comienzo de una etiqueta con sus atributos,
- *characters()* con el contenido de la etiqueta,
- y *endElement()* cuando encuentra el cierre de la etiqueta.

La clase *Parsejador* tiene un método llamado *reset()* que se llama, antes de parsear, pasándole los dos punteros a las listas que debe rellenar con jugadores y partidos jugados según se vayan leyendo en el fichero XML que contiene la base de datos.

Un punto que hay que mencionar es el algoritmo empleado para generar los partidos. El código es bastante largo, así que solo veremos como se resume en los siguientes puntos:

1. primero se generan todas las posibles combinaciones entre jugadores presentes descartando los partidos que ya se han jugado;
2. se evalúa cada combinación teniendo en cuenta las veces que ha jugado cada participante (individualmente y en conjunto) y la diferencia con el partido anterior (para que no se repitan los jugadores) asignándoles una puntuación;
3. la combinación con mejor puntuación se pasa a la lista de partidos a jugar y se actualizan los contadores de las veces que a jugado cada jugador;
4. se vuelve al punto 2 hasta que tengamos completa la lista de 15 partidos a jugar.

# 7 – Pruebas

Las pruebas inmediatas durante la fase de implementación se hicieron en el explorador Mozilla Firefox, que facilita el trabajo con el complemento Firebug que se comenta en el apartado *5.2 - Herramientas*. Posteriormente, en fase de pruebas, se ha visualizado la aplicación en varios exploradores. Se han tenido que probar formas alternativas de implementación de ciertos detalles para que salieran bien en todos los navegadores.

Se ha chequeado la aplicación con varias herramientas que analizan el código en línea validando su corrección y que cumple con los estándares. En concreto se han utilizado los validadores del consorcio W3C (World Wide Web Consortium):

- http://validator.w3.org para validar el código XHTML
- http://jigsaw.w3.org/css-validator/ para validar el código CSS
- http://validator.w3.org/checklink para comprobar que no haya enlaces rotos

Al realizar los test se corrigieron algunos enlaces a páginas que se habían movido debido a una reestructuración del código. Tras dichas correcciones, la aplicación pasa todos los chequeos satisfactoriamente. En la figuras 35, 36 y 37 se han capturado las pantallas de resumen del resultado de cada test para la página de entrada del portal.

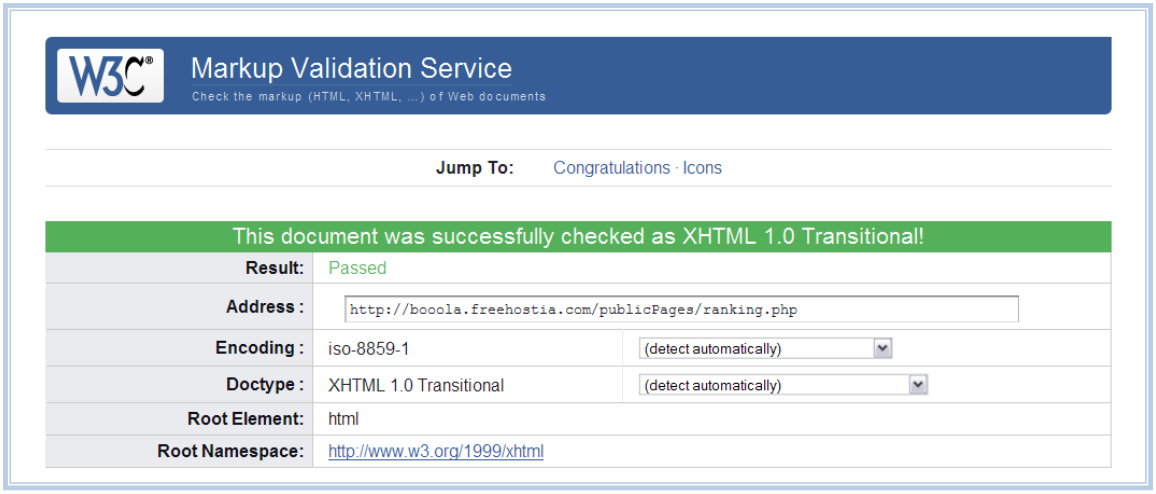


*Figura 35: Resumen del validador de marcas XHTML*


*Figura 36: Resumen del validador de CSS*

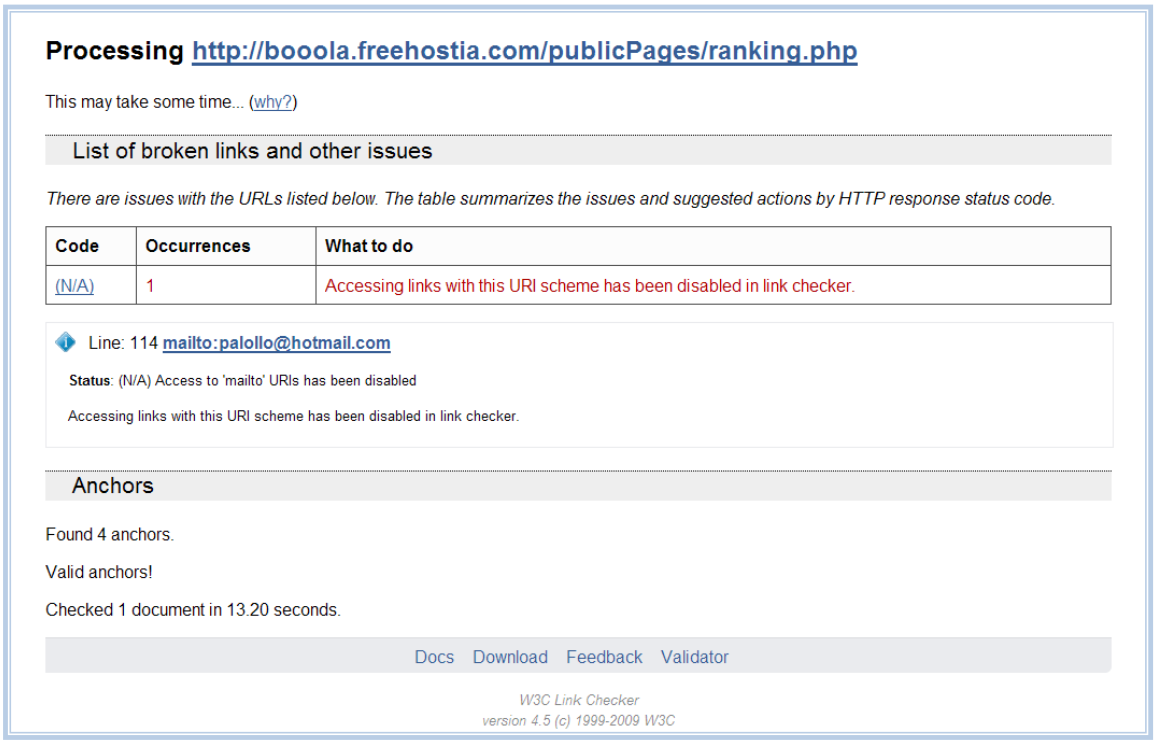


*Figura 37: Resumen del validador de enlaces*

En la figura 37 se observa que falla la comprobación del link a la dirección de correo electrónico, esto es porque no enlaza con otra página. Sin embargo, el link es correcto ya que lo interpretará el navegador, que en lugar de acceder a otra página reconoce el protocolo *mailto* y abrirá el gestor de correo predeterminado del usuario para enviar el correo.

Se ha comprobado también que la interfaz no se deforma al cambiar entre diferentes resoluciones. Cuando la página no cabe en la ventana del explorador aparecen barras de desplazamiento (vertical y/o horizontal) que el usuario debe usar para ver el resto de la página. Cuando la ventana es más grande que la página, esta última aparece centrada dentro de la ventana y el resto se rellena con el color del fondo.

Se ha probado durante una liga real. Fruto de las pruebas han sido numerosas ampliaciones que se incluyeron en su día y que ya se han mencionado en los capítulos anteriores. Ejemplos son: la paginación del foro, la posibilidad de insertar componentes en el mensaje de los eventos del calendario, diversos errores que se han ido corrigiendo y validaciones que se han añadido en campos de formularios.

## 7.1 – Pruebas de GesPartidos

Se ha probado el asistente en varios dispositivos de distintas marcas. El mayor problema fue el acceso al sistema de ficheros.

En primer lugar, en cuanto cambias de marca, cambia la estructura de directorios y ya no existe el directorio en el que debería estar por defecto el fichero de entrada, que contiene la base de datos de la liga. En ese caso, además, no siempre se pueden crear directorios libremente donde se quiera.

Y en segundo lugar, la estructura de directorios que uno ve cuando accede al dispositivo a través del PC, muchas veces, no concuerda con el que se ve desde la aplicación. A menudo suele ocurrir que desde la aplicación se ve solo una parte del sistema de ficheros global del dispositivo ya que las aplicaciones J2ME se ejecutan en un entorno restringido, al que muchos llaman “sandbox”.

Para resolver el primer inconveniente y ayudar en el segundo (ya que al final uno acaba conociendo su dispositivo) se implementó un pequeño explorador para buscar el fichero de entrada del asistente tal como se explica en el apartado *5.3.1 - Implementación de GesPartidos*.

Durante las muchas pruebas que se realizaron se detectó también que el mecanismo de control de permisos de las aplicaciones J2ME es bastante molesto en ciertas ocasiones. Funciona del siguiente modo:

El acceso a los distintos recursos del dispositivo se controla mediante permisos en la máquina virtual para cada aplicación o para todas en general (según el dispositivo). En este caso solo nos interesa uno: el permiso de lectura y escritura en ficheros. Por de defecto, la máquina virtual suele estar configurada para que pregunte al usuario justo antes de conceder un permiso. Y no hay forma de configurarla para que los conceda directamente sin preguntar. De esta forma, el sistema preguntará al usuario nada más arrancar la aplicación si puede leer del sistema de ficheros para acceder al fichero de entrada y, si no se encuentra el fichero y se decide buscarlo, durante la exploración irá preguntando si se le concede el permiso cada vez que el usuario a acceda a un nuevo directorio.

Esto puede resultar molesto, sin más. Para evitarlo, se puede firmar la aplicación con una firma electrónica que nos proporcione una autoridad certificadora cuyo certificado esté instalado en el dispositivo. Cuando un *midlet* está firmado, se le conceden los permisos sin preguntar (según el nivel de acceso que le confiera la firma con que se ha firmado). Las autoridades certificadoras cuyos certificados se instalan en los móviles no suelen regalar firmas electrónicas, así que se ha desestimado esta opción. Sin embargo el procedimiento para firmar un *midlet* se incluye en uno de los anexos de este documento.

# 8 – Conclusiones

Empecé a desarrollar este proyecto por iniciativa propia, con la simple idea de aprender una serie de tecnologías que no me enseñaron en la carrera, que se emplean muchísimo en el mundo profesional y que personalmente me atraen. Quería desarrollar un proyecto que cubriera una necesidad en aquel momento o al menos que lo pudiera aprovechar. Y entonces vi que las tecnologías web son lo más idóneo que se puede aplicar en una liga para cualquier deporte: con el mismo sistema puedes gestionar la liga a la vez que publicas los resultados y le das publicidad. Entonces organicé una liguilla de frontenis entre amigos para probar la aplicación conforme la desarrollaba. Y fruto de la necesidad, posteriormente surgió la idea de implementar una herramienta auxiliar que pudiera ejecutar en un dispositivo móvil para gestionar los partidos en la misma pista de juego. De hecho era otra de mis inquietudes remanentes: averiguar cómo realizar una aplicación que pudiera ejecutar en cualquier parte desde mi teléfono móvil.

Tuve que documentarme bastante, buscando la información por mi cuenta, al fin y al cabo casi todo está en Internet, pero hay que saber encontrarlo.

El desarrollo fue progresivo: primero se implementó algo básico y posteriormente se fueron añadiendo nuevos requisitos y mejorando las funcionalidades. Como empezó siendo un proyecto personal no había plazos aunque ciertas funcionalidades sí requerían estar terminadas para poder llevar al día la gestión de la liga que desempañábamos.

Al ser un proyecto formativo, siempre traté de hacer las cosas bien, siguiendo las pautas y la forma de trabajar que me ensañaron en la carrera: separando en capas, modularizando, reutilizando código y documentando cuanto más mejor.

Creo que al final ha quedado un buen producto, bien implementado y más o menos completo, que hace uso de muchas y diversas tecnologías para llevar a cabo la gestión integral de una liga, aunque sea pequeña y con unas reglas de funcionamiento un tanto peculiares.

Como posible ampliación se propone, como se ha comentado ya, la creación de nuevos “temas” para otros deportes que cambien el fondo y la apariencia de la web (los estilos CSS). Sería conveniente incluir la selección del tema como opción en el fichero de configuración.

Otra ampliación que puede ser interesante sería portar el asistente GesPartidos a otras plataformas para móviles como Android, Symbian o iPhone OS.

# 9 – Bibliografía


- Félix Buendía García.
  *Una guía para la realización y supervisión de proyectos final de carrera (PFC) en el ámbito de la web*.
  Editorial Universidad Politécnica de Valencia, 2008.

- Lidia García Esparcia, Helena Saura Martín y Silvia Silla Juan.
  *Sitio web de un polideportivo*
  Proyecto final de carrera, ETSIA-UPV, 1994.

- José Carsí, Juan Sanchez y Javier Jaén.
  Apuntes para la asignatura de "*Ingeniería de la programación*".
  FIV-UPV, curso 06-07.

- Manual de PHP online.
  http://www.php.net/manual/en/
  1997-2009. Accedido en 2009.

- W3C
  *XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition)*
  January 2000, revised August 2002.
  http://www.w3.org/TR/xhtml1/

- W3schools.com
  *HTML 4.01 / XHTML 1.0 Reference*.
  http://www.w3schools.com/tags/default.asp

- Sergio Sáez Barona.
  *Introducción a la programación de sitios web dinámicos con PHP y MySQL*.
  Curso de PHP y MySQL. Edición 2006.

- Manual de MySQL online.
  http://dev.mysql.com/doc/refman/4.1/en/
  2008-2010. Accedido en 2009.

- Sitio oficial de J2ME.
  http://java.sun.com/javame/index.jsp
  Accedido en 2010.

- Francisco Martínez Álvarez
  *Desarrollo de aplicaciones JAVA para móviles: J2ME y herramientas de desarrollo.*
  Departamento de Lenguajes y Sistemas Informáticos. Cursos de Doctorado.
  Universidad de Sevilla. Junio de 2006.
  http://www.lsi.us.es/docencia/get.php?id=1402

- F. Javier García Castellano
  *Introducción a MIDP 2.0*
  http://leo.ugr.es/J2ME/MIDP/intro.htm

- Portal web del grupo de investigación J2ME-grasia
  http://grasia.fdi.ucm.es/j2me/_J2METech/index.html
  Universidad Complutense de Madrid. 2004.

- Portal Wikipedia.org
  http://es.wikipedia.org/wiki/Http
  http://es.wikipedia.org/wiki/Html
  http://es.wikipedia.org/wiki/Xhtml
  http://es.wikipedia.org/wiki/Programacion_por_capas
  http://es.wikipedia.org/wiki/XML
  http://es.wikipedia.org/wiki/SQL

# 10 – Anexos

## 10.1 – Sentencias SQL para la creación de las tablas de la BD

A continuación se muestran las sentencias en SQL para MySQL que deben usarse para crear las tablas que usará la aplicación en la base de datos.

```
drop table if exists partit;
drop table if exists equip;
drop table if exists jugador;
drop table if exists calendari;
drop table if exists foro;

create table jugador(
id integer not null auto_increment,
nom varchar(50) not null,
guanyats integer not null,
perduts integer not null,
pilotades integer not null,
valid boolean not null,
constraint cp_jugador primary key(id),
constraint un_jugador unique(nom) );

create table equip(
jug1 integer not null,
jug2 integer not null,
companys integer not null,
contrincants integer not null,
constraint cp_equip primary key(jug1, jug2),
constraint ca_equip_jugador1 foreign key(jug1) references jugador(id),
constraint ca_equip_jugador2 foreign key(jug2) references jugador(id) );

create table partit(
num integer not null auto_increment,
dia date not null,
jug1 integer not null,
jug2 integer not null,
jug3 integer not null,
jug4 integer not null,
resultatEqA integer not null,
resultatEqB integer not null,
valid boolean not null,
constraint cp_partit primary key(num),
constraint un_partit unique(jug1, jug2, jug3, jug4),
constraint ca_partit_jugador1 foreign key(jug1) references jugador(id),
constraint ca_partit_jugador2 foreign key(jug2) references jugador(id),
constraint ca_partit_jugador3 foreign key(jug3) references jugador(id),
constraint ca_partit_jugador4 foreign key(jug4) references jugador(id) );
```

```
create table calendari(
id integer not null auto_increment,
tipo enum('remarcat', 'normal', 'passat', 'separador') default 'normal',
dia tinytext,
event text not null,
constraint cp_calendari primary key(id) );

create table foro(
id integer not null auto_increment,
nick tinytext not null,
missatge text not null,
fecha datetime not null,
ip varchar(20),
valid boolean not null default true,
constraint cp_foro primary key(id) );
```

## 10.2 – Cómo proteger el directorio de administración en Apache

Para evitar que nadie pueda acceder al menú de administración excepto los administradores se emplea la protección con autentificación que proporcionan los servidores web. En caso de contratar este servicio mediante alguna empresa de *hosting* en Internet, se suele poder configurar esta opción fácil y cómodamente desde el menú de administración del sitio contratado. En otro caso hay que contactar con el administrador del servidor web.

En este anexo se explica la forma de proteger un directorio en un servidor web Apache. Es decir, lo que tenemos que hacer si somos nosotros mismos los administradores del servidor. Vamos a usar como ejemplo el servidor que se ha instalado para las pruebas de desarrollo con el paquete Xampp. Sin embargo, el procedimiento se explica de forma genérica y es completamente análogo para cualquier instalación de Apache que tengamos.

La protección funciona del siguiente modo:

Cuando el usuario intenta acceder a alguno de los recursos que se encuentran dentro del directorio protegido, el servidor solicita un nombre y una contraseña para obtener la identidad del usuario. Si los datos introducidos corresponden a un usuario válido (en este caso, un administrador de GesLigas), el servidor enviará el recurso (la página del menú de administración), en otro caso se denegará el servicio.

Hay dos tipos de autenticación: *basic* y *digest*.

A continuación se explica que características tiene cada una de las dos opciones y cómo se configura el servidor para usar una u otra.

### Protección tipo *Basic*

Esta es la protección básica. Es sencilla y no requiere de ningún módulo extra sobre Apache.

Existe un fichero donde el administrador registra los usuarios que deben tener acceso a los recursos (cada usuario con su contraseña). Cuando un usuario solicita un recurso protegido enviando sus credenciales, el servidor comprueba si existe alguno registrado que case con este y, consecuentemente sirve el recurso o lo deniega.

Para generar el fichero donde se registran los usuarios, se utiliza la herramienta *htpasswd.exe* que se encuentra en el subdirectorio *bin* dentro del directorio donde esté instalado Apache. Desde un terminal de comandos (o "símbolo del sistema") se ejecuta con los siguientes parámetros:

```
$> htpasswd.exe -c C:\xampp\apache\conf\.htpasswd adminGesLigas
```

Con esta orden se creará el fichero *C:\xampp\apache\conf\.htpasswd* con un usuario llamado "adminGesLigas" que llevará asociada la contraseña que nos pregunta la herramienta tras ejecutar la orden. Con el parámetro "-c" se genera un nuevo fichero de usuarios borrando el

de antes si lo hubiera. Si solo queremos añadir un usuario al fichero ya creado, omitiremos el parámetro. Siempre se recomienda crear el fichero de usuarios fuera del directorio de recursos que se sirven.

Lo siguiente es configurar Apache para que consulte los ficheros *.htaccess* que puede haber en cada directorio y que determinan la protección del directorio.

Buscamos el fichero de configuración de Apache y lo editamos para que la directiva "AllowOverride" esté configurada como "All".

C:\xampp\apache\conf\httpd.conf

```
[…]
    #
    # AllowOverride controls what directives may be placed in .htaccess files.
    # It can be "All", "None", or any combination of the keywords:
    #   Options FileInfo AuthConfig Limit
    #
    AllowOverride All

[…]
```

Ya tan solo queda crear un fichero .htaccess en los directorios que se quieran proteger. Por ejemplo, para proteger el directorio *publicPages* de GesLigas en nuestro servidor de Xampp crearemos el siguiente fichero.

C:\xampp\htdocs\GesLigas\adminPages\.htaccess

```
AuthType Basic
AuthName "acceso restringido"
AuthUserFile C:\xampp\apache\conf\.htpasswd
Require user adminGesLigas
```

La primera línea indica el tipo de autenticación. La siguiente indica el mensaje que mostrará el navegador al solicitar la contraseña. Con *AuthUserFile* indicamos dónde se encuentra el fichero de usuarios que se debe consultar. Y con *Require user* establecemos el subconjunto de usuarios (separados por espacios en blanco) que pueden acceder al directorio.

Truco:
Si el explorador de Windows no permite crear ficheros con nombres que empiecen por punto, se puede emplear un terminal de comandos, por ejemplo con la orden *edit*:

```
$> edit .htaccess
```

### Protección tipo *Digest*

Con el tipo de protección básica, la contraseña viaja por la red sin cifrar. De tal forma que puede ser obtenida por un usuario malintencionado simplemente con "escuchar" la red por la que circule la información mediante un *sniffer* (herramienta que captura y analiza el tráfico de la red). Hay que tener en cuenta que la contraseña se transmite cada vez que el navegador cambia de página, es decir, se requiere la autenticación con cada recurso que se solicita al servidor, solo que el usuario no tiene que volver a introducir sus credenciales ya que el navegador los "recuerda" y los envía automáticamente cada vez.

Si usamos autenticación de tipo *digest* la contraseña se cifra con MD5 (un algoritmo criptográfico de 128 bits) antes de ser enviada por la red. La especificación completa de la autenticación *digest* se puede consultar en el estándar RFC 2617.

Este tipo de autenticación puede no estar soportada en algunos navegadores antiguos.

Para poder usar el tipo *digest* hay que modificar el fichero de configuración de Apache para que se cargue el módulo de autenticación *digest* cada vez que arranque el servidor. En concreto hay que descomentar la siguiente línea:

C:\xampp\apache\conf\httpd.conf

```
[…]
LoadModule auth_digest_module modules/mod_auth_digest.so
[…]
```

Para que se cargue el módulo hay que reiniciar el servidor.

La autenticación *digest* requiere un fichero de usuarios diferente al de la autenticación básica, que se crea con otra herramienta llamada *htdigest.exe* (en el subdirectorio *bin* de Apache) y que funciona de la misma forma que *htpasswd.exe*. Para crear el fichero de usuarios ejecutaremos la siguiente orden en el terminal de comandos:

```
$> htdigest.exe -c C:\xampp\apache\conf\digest "acceso restringido" adminGesLigas
```

Esto nos preguntará la contraseña para el nuevo usuario "adminGesLigas", que debemos introducir y confirmar. Recordemos que, si solo queremos añadir un usuario al fichero de usuarios ya creado, se debe omitir el parámetro "-c". Obsérvese un nuevo parámetro justo antes del nombre del usuario, con el cual indicamos tan solo un nombre para el acceso que debe coincidir con el valor de la directiva *AuthName* en el fichero *.htaccess*.

Lo último que queda es, modificar el fichero *.htaccess* en el directorio de administración de GesLigas, que ahora debería contener lo siguiente:

C:\xampp\htdocs\GesLigas\adminPages\.htaccess

```
AuthType Digest
AuthName "acceso restringido"
AuthDigestProvider file
AuthUserFile C:\xampp\apache\conf\digest
Require user adminGesLigas
```

## 10.3 – Cómo firmar un *midlet*

En este anexo se explica cómo firmar midlets con NetBeans mediante una firma electrónica que nos haya proporcionado una autoridad certificadora. A continuación se explica también cómo generar tu propia firma y certificado. Esto nos servirá si podemos instalar el certificado en el dispositivo (en ninguno de los móviles que he visto se podía hacer; no hay que confundir los certificados para la web, con los de la maquina virtual java, normalmente se instalan en sitios distintos).

Lo primero es bastante sencillo, NetBeans lo facilita mucho pero hay que saber lo que se está haciendo para que todo funcione correctamente.

En el panel lateral de "proyectos", pinchamos con el botón derecho del ratón en nuestro proyecto y accedemos a las propiedades del proyecto. Desplegamos la categoría "Build" y seleccionamos "Signing" tal como se muestra en la figura 38.

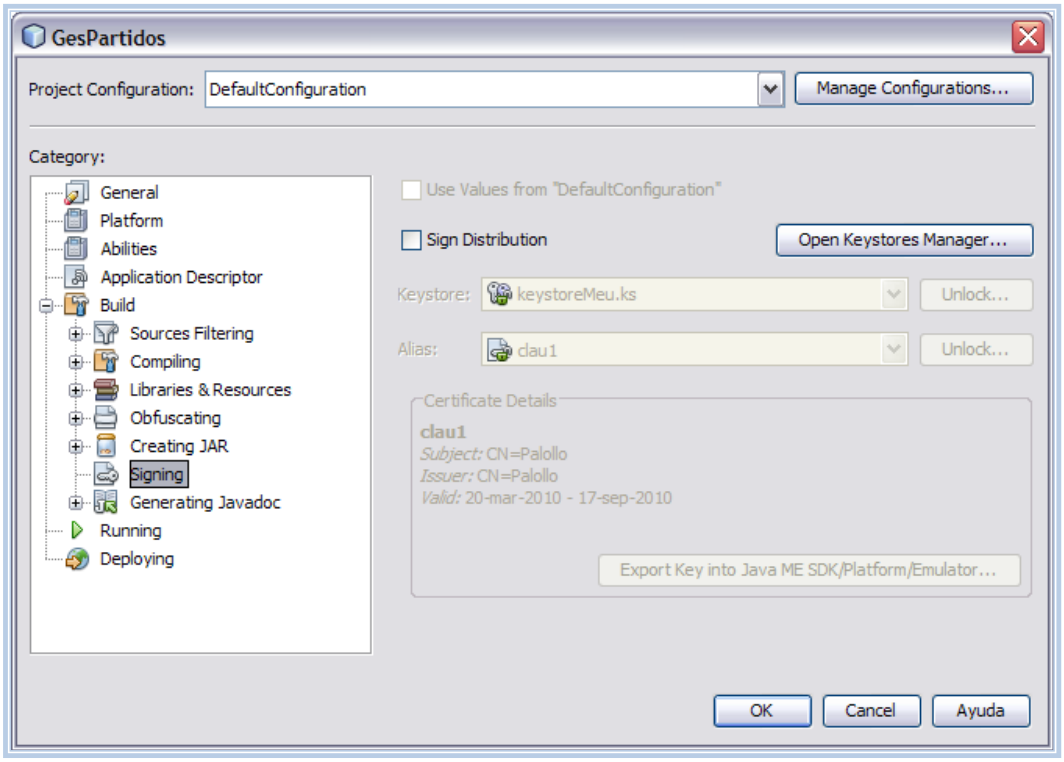


*Figura 38: Ventana de propiedades del proyecto*

Aquí marcamos la opción "Sign Distribution". Tendremos que importar la firma que nos haya proporcionado la autoridad certificadora, que suele estar contenida en un fichero con formato PKCS #12 (con extensión .p12). Para importar pulsamos en el botón "Open Keystores Manager", luego en "Add Keystore", marcamos "Add existing Keystore" y seleccionamos el fichero. Cerramos el *Keystore Manager* y seleccionamos la firma desbloqueando con el password correspondiente. Y eso es todo, la próxima vez que hagamos un Build del proyecto obtendremos el fichero *.jar* firmado en el subdirectorio *dist* del directorio de proyecto.

### Generar tu propia firma y certificado

El procedimiento para generar tu propia firma es tan sencillo como el anterior. Tan solo hay que crear un nuevo *keystore* (o almacén de llaves) en lugar de importar uno existente, tal como podemos ver en la figura 39.

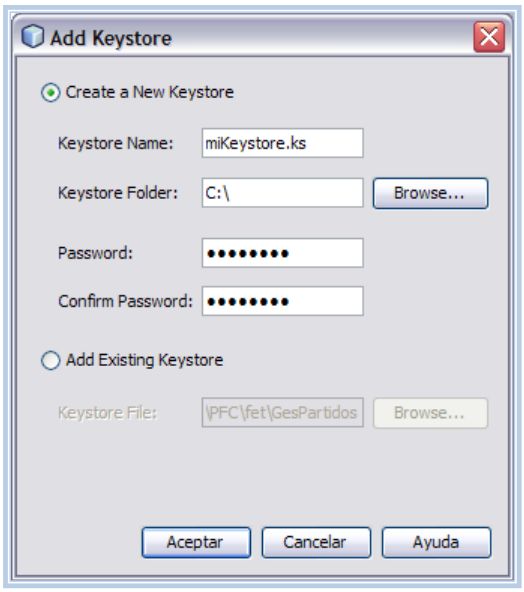


*Figura 39: Ventana para añadir keystores*

Le damos un nombre, un directorio donde se guardará y el *password* con el que se cifrarán las firmas o claves que contenga.

Tras crear el *keystore*, hay que añadir algún *key pair* (un par de llaves, publica y secreta, que vienen a ser la firma). En el *Keystores Manager* seleccionamos el *keystore* que acabamos de crear y pulsamos en "New…".

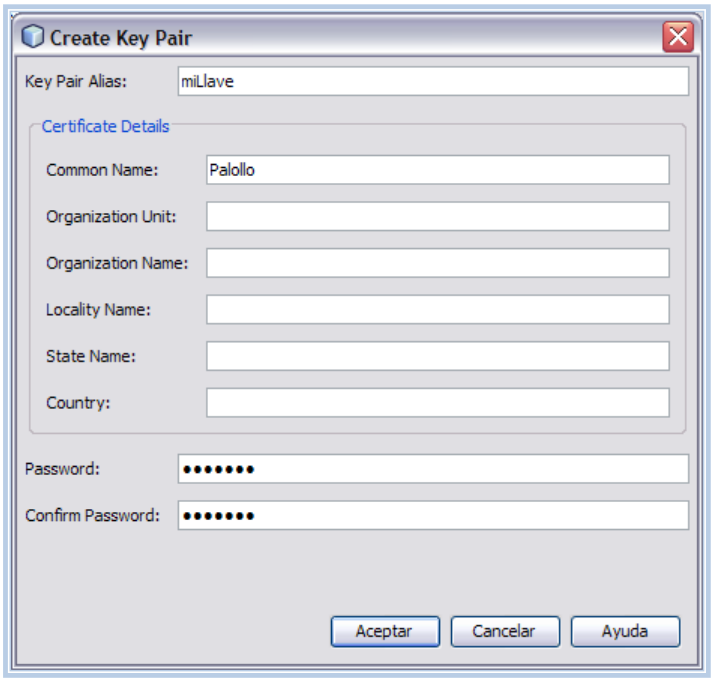


*Figura 40: Ventana para crear un par de llaves*

Escribimos un alias para identificarla, rellenamos al menos uno de los campos de detalles del certificado e introducimos una contraseña con la que se cifra el par de llaves.

Bien, ahora como antes, cerramos el *Keystore Manager* y seleccionamos el *keystore* y la llave que acabamos de crear. Y así generaremos el *midlet* firmado con nuestra propia llave.

Pero para que sea reconocida por el dispositivo hay que instalar el certificado en él y para extraer el certificado recurrimos a las herramientas del JDK (Java Development Kit) en ventana de comandos para trabajar con claves.

Si tenemos NetBeans, tenemos JDK (es un requisito). Así que lo buscamos y accedemos al subdirectorio *bin*, allí se encuentra la herramienta *keytool.exe* que nos servirá para nuestro propósito. Lo ejecutamos desde un terminal de comandos (o "símbolo del sistema") con los parámetros siguientes:

```
$> keytool.exe -exportcert -v -alias miLlave -file C:\miCertificado.cer -keystore C:\miKeystore.ks
```

La herramienta preguntará por la contraseña del almacén de llaves y generará el certificado en *C:\miCertificado.cer*.