4618:

DESARROLLO DE UN PORTAL WEB PARA INSTITUCIONES DEPORTIVAS UTILIZANDO COMPONENTES OPEN SOURCE

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
Xavier Carrillo Navinés
i dirigit per
Diego Javier Mostaccio Mancini
Bellaterra, 19 de Juny de 2012

Índice

1 Introducción

1.1 Contexto del proyecto

Este es un proyecto en el que se busca analizar una herramienta de creación de portales orientado a una temática concreta. Por esto se ha buscado la colaboración externa de personas, en principio, ajenas al proyecto y que tienen conocimientos del campo en cuestión, pudiendo ofrecer una visión más cercana.

La tarea de estas personas es la de hacer el papel de un cliente que requiere que desarrollen una aplicación. Intentando tener unos requerimientos realistas con los que poder sopesar los pros y los contras de la herramienta de desarrollo en cuestión.

También es interesante comentar el hecho de que se busca solo trabajar con material *Open Source*.

1.2 Motivaciones

Actualmente se puede afirmar que tener presencia en la Red es prácticamente equivalente a existir en un sector. Esto hace que se vuelque mucho esfuerzo y recursos en el desarrollo en el campo de los portales web.

Generalmente muchos usuarios han trabajado (o lo han intentado) en la creación de material para la Red (o con un formato capaz de acabar en esta). Pero se trata de un mundo que recibe grandes dosis de trabajo y recursos. Esto hace que las tecnologías dedicadas a su desarrollo avancen o cambien a gran velocidad y cada vez hay más y mejores herramientas para trabajar. Es interesante buscar un sector en el que centrarse con el objetivo de ver que ofrece la comunidad en cuanto a desarrollar portales.

Con la crisis en la que vivimos cobra una gran envergadura el deporte. Este proporciona una forma de entretenimiento tanto para el que lo practica como para el público que lo sigue y además aporta buenos valores a la sociedad: trabajo en equipo, competitividad, sentimiento patriótico, superación, salud. Todo esto ligado a la gran cantidad de dinero que mueve, convierte el mundo del deporte en un campo de trabajo ideal.

Puesto que existen muchas tecnologías de cara a la creación de portales web (*HTML*, *PHP*, *Javascript*, *JSPs*, etc.) es interesante estudiar una herramienta que pueda manejarse con la mayoría (o todos) ellas. El uso de herramientas *Open Source* le da un mayor interés ya que puede disminuir de forma considerable el coste.

1.3 Objetivos

Se llevará a cabo un análisis de herramientas *Open Source* de desarrollo de portales web teniendo presente la competitividad de estas. Para hacer esto se tendrán en cuenta distintos factores:

- Se trabaja sobre un sector determinado y acotado.
- Se contemplarán ideas propias sobre este sector.
- Se busca la opinión de una entidad del sector.

Esta entidad proporcionará unos requerimientos como si de un cliente real se tratara de forma que también sirva como experiencia de trabajo con clientes y diseño de aplicaciones.

Así pues marcamos como principales objetivos:

1. Análisis de herramienta de creación de portales web.
2. Diseño de un portal con requisitos reales e impuestos por un cliente.
3. Interactuar con la comunidad de herramientas *Open Source*.
4. Hacer un prototipo de portal.

1.4 Estructura de la memoria

A parte de la introducción y las conclusiones finales la memoria consta de otros cuatro capítulos: ‘Requerimientos y necesidades del portal’, ‘Estado del arte y elección de herramienta’, ‘Diseño’ y ‘Análisis’.

Requerimientos y necesidades del portal: explica básicamente el porqué de un cliente y que información se recabó y como se logro acotar el problema en las reuniones.

Estado del arte y elección de herramienta: un vistazo al mundo de los portales deportivos y alagunas de las distintas herramientas *Open Source* que podrían ser de utilidad. Explicación del porqué se ha elegido analizar *Liferay*.

Diseño: Soluciones propuestas a los contenidos desglosados en el capítulo de 'requerimientos y necesidades del portal'.

Análisis: Módulos, funcionamiento y posibilidades de la herramienta. Como puede aplicarse para solucionar el problema planteado.

1.5 Planificación

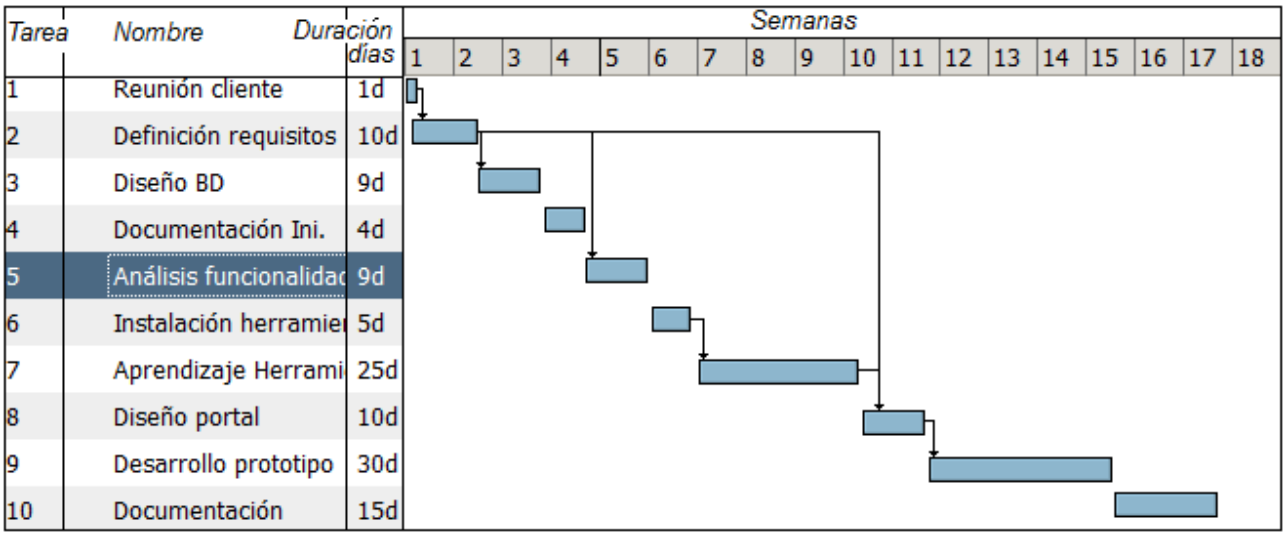

Figura 1.1 – Diagrama de Gantt.

La planificación cuenta de aproximadamente unos 118 días (cuatro meses) sin contar festivos y finales de semana. Las tareas son generalistas y muchas de ellas (especialmente las de análisis y desarrollo) se podrían dividir en múltiples tareas. La tarea de documentación inicial consta en la redacción del informe previo y una hoja de requerimientos (anexo 7.1).

2 Requerimientos y necesidades del portal

El cliente antes mencionado es un club de atletismo que abarca atletas de diversos municipios y tiene una sección infantil donde entrenan niños.

2.1 Adquisición de requisitos

Un portal deportivo tiene un gran abanico de posibilidades (o en este caso necesidades) según el nivel de la entidad, el tipo de deporte o deportes a los que se dedica, la variedad de socios o afiliados, el perfil o el número de estos (la masa social) y muchos otros factores a tener en cuenta.

Esto añade importancia al tener una entidad deportiva en la que acotar el trabajo. Centrando el problema en un caso concreto pero con la intención de no perder de vista la idea de analizar hasta qué punto se pueden generalizar las funcionalidades para cualquier tipo de portal deportivo.

El contar con una entidad que proporciona unos requisitos reales, ajustados su forma de dirigir un club deportivo, permite tener una visión más real (en contrapartida a la propia) de lo que un cliente espera.

Con este primer objetivo en mente se programa una reunión con dos representantes de la entidad en cuestión. Uno de ellos cuenta con algo de experiencia en el desarrollo de los portales suavizando la comunicación con el segundo, que cuenta con un perfil más cercano al de un usuario estándar.

Ambos aportan una visión distinta y el hecho de contar con un cliente con conocimientos de este mundo ayuda a delimitar las necesidades de datos para el portal. Por otro lado con la opinión de alguien sin conocimientos de desarrollo de portales ayuda con las necesidades y preferencias de un usuario estándar que quizás presta más atención a otros aspectos que en principio pueden parecer de poca importancia y que son de interés para los usuarios finales del portal.

Contar con estos dos perfiles de cliente agiliza la separación de funcionalidades y necesidades que se precisan en el portal por el lado del usuario (*front-end*) y por el lado del administrador (*back-end*).

En la reunión se lleva a cabo un trabajo de toma de datos, se comentan las distintas visiones y se

proponen algunas ideas y soluciones a varias peticiones. Tras esto la comunicación se mantiene de forma remota vía correo electrónico.

Se genera una hoja de requerimientos con lo que se ha acordado y se presenta al cliente. Este a su vez proporciona un *feedback* de modo que se llega a tener un documento consensuado por el cliente de los requerimientos.

2.2 Necesidades y propuestas del cliente

En desarrollo web podemos distinguir según la visualización del usuario que navega por un portal, *Front-end*, y el administrador del sitio, *Back-end*. Así pues también dividimos las necesidades en estos dos grupos.

Front-end

Una de las primeras consideraciones que dio el cliente es la de tener en cuenta que cualquier visitante del portal se consideraría un usuario, permitiéndole navegar por las secciones y noticias, visualizar información pública del club (accesible por cualquier miembro que no cuente con algún papel administrativo dentro del portal) y participar en la comunidad, principalmente con comentarios.

El portal además se quería adaptar a diversos idiomas, en principio catalán, castellano e inglés.

La entidad tiene distintas secciones diferenciadas por modalidad y edad. La única sección separada por edad es la de 'Escuela' y se tiene en cuenta que los integrantes de esta categoría son menores y deben tratarse de forma distinta por consideraciones tanto legales como de información (un menor no tiene porqué contar con un NIF).

Las secciones regidas por modalidad lo son por el tipo de superficie en la que se practica el deporte. El club las identifica como 'Pista', 'Carretera' y 'Montaña'. Junto a 'Escuela' son las cuatro secciones que se requieren en el portal. Estas no son más que páginas de noticias sobre cada modalidad dentro del club. A parte de las categorías de la entidad el portal necesitaría otras secciones, detalladas a continuación.

La página principal es también una página de noticias donde aparecen noticias generales, anuncios de la entidad a sus visitantes y también las últimas noticias de las secciones permitiendo así a un visitante asiduo estar informado sin la necesidad de navegar por todo el portal.

Por motivos de estética y comodidad la página principal solo mostraría las noticias más recientes y algunos avisos o mensajes del club o el administrador (una felicitación de las fiestas, aniversario del club). Si se desea leer noticias antiguas se pueda hacer desde una sección del portal llamada 'Histórico' con un formato de estilo blog.

Además el club cuenta con patrocinadores. Para estos y para los atletas interesan dos secciones diferenciadas en la web donde mostrar información sobre estos. En el caso de patrocinadores mostrando un logo, el nombre, que tipo de entidad es y alguna información más (por ejemplo la dirección de la sede). En el caso de los atletas del club se quiere mostrar una foto acompañada de nombre, apellidos y algunas marcas personales de tiempo en carrera.

A parte de los entrenamientos el club también organiza sus propios eventos y carreras. Las 'Carreras' deben contar con una sección en el portal y debe permitir no solo visualizar estos eventos sino también inscribirse ya sea como participante o como voluntario. En este apartado al tratarse de un club deportivo se fomenta la competitividad y se desea poder ver quien esta apuntado para participar en las carreras.

Finalmente la sección de contacto la cual muestre algo de historia del club, la sede social y datos de contacto. Quedando dividido el portal en diez secciones: 'Inicio', 'Patrocinadores', 'Carreras', 'Contactar', 'Histórico', 'Atletas', 'Pista', 'Carretera', 'Montaña' y 'Escuela'.

Como se ha comentado todo visitante del portal será considerado usuario pero también se prevé la necesidad de un sistema de registro y autenticación. De esta forma se podrán registrar nuevos socios en línea y a los que ya sean socios les da la opción de participar de forma no anónima en el portal.

A pesar de que todo usuario puede navegar por el portal en el momento en el que se quiera participar en un evento, ya sea como corredor o como voluntario (se forme parte o no del club) se requerirán unos datos reales para inscribirse y presumiblemente el pago de una cuota.

Back-end

En lo referente a la administración se tiene que poder moderar comentarios, crear y editar noticias y eventos (principalmente carreras), administrar altas y bajas tanto de patrocinadores como de socios o acceder a algunos criterios de configuración del portal.

A parte de estas tareas de administración se espera que pueda generar distintos informes con información que recoge el portal o la base de datos del club. Básicamente activos del club, participantes en eventos o información detallada según fechas, además de poder exportar estos informes en un formato legible.

2.3 Consideraciones

Hay que tener en cuenta que estos requerimientos son para un ejemplo de portal en concreto pero se busca la posibilidad de generalizar la idea al máximo y analizar las posibilidades a un nivel de abstracción más alto.

Una primera consideración a tener en cuenta es la distinción entre usuarios. En los requerimientos del cliente ya aparece esta necesidad al tener que diferenciar entre tipos de usuarios según su edad o en el caso de participar en un evento la distinción entre los socios y los no socios. De todos modos esta diferenciación de usuarios no aporta una jerarquía.

En un portal en general, sea o no deportivo, la jerarquía de los usuarios es importante y esto es algo que el cliente no especifica o no ve necesario.

En el portal de una empresa los usuarios deberán tratarse de forma distinta según su estatus dentro del organigrama. Generalizando podríamos encontrarnos con un portal que diferencia a los socios de los entrenadores dándoles algunos privilegios los unos sobre los otros o acceso a distintos recursos (un ejemplo paralelo a este sería en el caso de un portal académico en la diferenciación de alumnos y profesores).

Además se pude necesitar o querer una agrupación de usuarios según un criterio aunque estos no sean distintos entre sí. En el caso de socios del club agruparlos por criterios como compartir unas instalaciones o un entrenador (o en el ejemplo académico forman parte del mismo grupo o clase).

De cara a la organización de eventos un portal deportivo puede tener la necesidad de que los eventos sean privados (solo accesibles para los socios) o incluso la posibilidad de poder crear eventos de los dos tipos: públicos y privados. En este caso la jerarquización y la posibilidad de agrupación de usuarios sería una buena forma de encarar esta funcionalidad.

Otra consideración a tener en cuenta es que el cliente como club deportivo presta un servicio abierto al cual puede acceder cualquier persona. No todas las entidades deportivas funcionan de esta forma. Existen entidades mucho más restrictivas y que su portal no pasa de ser una herramienta informativa y/o de marketing o publicidad. De todos modos a parte de considerar esta opción no parece añadir funcionalidad de interés más allá de integrar una tienda o venta de entradas.

Finalmente comentar que tener un portal no deja de ser una forma de crear o unificar una comunidad y aunque no se han requerido por el cliente en este caso se podría pensar en añadir herramientas de interacción entre usuarios. Por ejemplo, se podría incluir un foro o un chat interno. En el caso de una entidad con un portal más enfocado a publicitarse que a prestar un servicio no sería tan interesante como en el de nuestro cliente.

3 Estado del arte y elección de herramientas

3.1 Estado del arte: Portales de entidades deportivas

Comparando los distintos portales deportivos, nos encontramos con una gran cantidad de entidades que distan mucho unas de otras. Esto dificulta el hacer una comparación y acotar el estado del arte. Existen entidades con presupuestos millonarios y secciones en todo tipo de deportes que cuentan con una masa social y un nivel de popularidad muy elevada (pudiendo abarcar visitas de los cinco continentes) mientras que hay otras entidades más modestas (municipales o de menor categoría) que también cuentan con algún tipo de portal deportivo.

Podemos encontrar, pues, desde el portal de un club de petanca [1] que podría estar desarrollado por un integrante del club o un individuo externo (no siempre conocimientos reales del sector y de los portales) a un club profesional con un equipo de marketing y desarrolladores asalariados que desarrollen un producto con unos fines muy distintos [2][3].

En el caso de asociaciones o clubs más modestos cada vez es menor la creación de portales ya que estos tienden al uso de redes sociales que permite tener presencia en la Red sin necesidad de hacer una inversión en la creación de un portal. De todos modos las entidades deportivas de mayor calibre también tienen presencia en este medio y normalmente en más de una red social a la vez.

En este aspecto también cabe remarcar la existencia de portales como los de las franquicias deportivas en los Estados Unidos, que no son clubs de la forma que se entienden en otras partes del mundo. Este modelo de franquicia enfrentado al de club se refleja en los portales que acaban siendo una mezcla entre una tienda y un portal deportivo.

Desde un punto de vista más estricto en el caso de las franquicias el portal es propiedad del organismo o asociación que organiza la competición (NBA [4], NHL [5], NFL [6]). Desde el mismo dominio accedemos a los portales de todas las franquicias afiliadas y sus secciones (equipos femeninos o equipos en ligas de desarrollo).

Comentado el caso de franquicias nos encontramos con el hecho de que las federaciones o asociaciones [7][8] que organizan torneos a nivel profesional también cuentan con sus propios portales. Estos divergen sensiblemente de los portales de clubs deportivos en el sentido más estricto

ya que no representan los intereses de una sola entidad pero siguen siendo portales deportivos. Al igual que en el caso de los portales de clubs, los distintos portales de asociaciones o federaciones pueden ser muy distintos entre ellos.

3.2 Estado del arte: Herramientas para la creación de portales

Actualmente los navegadores son una de las herramientas más potentes que existen, capaces de interpretar y ejecutar prácticamente cualquier cosa que se nos ocurra. En un principio con un simple documento *HTML* se mostraban páginas de texto con navegación. Actualmente cualquier usuario ve de lo más normal la reproducción de audio o video desde su navegador, portales con estéticas vistosas y elegantes y encontramos herramientas en línea capaces de hacer cualquier tarea, desde calcular series de Fourier [9] a videojuegos [10]. Esto es posible gracias, entre otras cosas, a la gran variedad de lenguajes y herramientas.

Prácticamente todos los desarrolladores importantes de Software tienen alguna herramienta dedicada o capaz de crear portales. En nuestro caso nos centraremos en repasar algunas herramientas de licencia de código abierto.

Los servidores pueden ejecutar aplicaciones, por ejemplo Java [11]. Con la ayuda de distintos *frameworks* y otros lenguajes dedicados a otras labores se puede implementar una gran variedad de funcionalidades para un portal, esto y la cada vez mayor cantidad de recursos lleva a la necesidad de creación de estándares. Unido a esto la posibilidad de reutilizar una funcionalidad y la atomización de las mismas hacen especialmente atractivos las herramientas que trabajan con *portlets* [12]. Encontramos diversas herramientas *Open Source* que trabajan con estos.

Como ejemplo encontraríamos ***OpenPortal*** [13]**, *uPortal*** [14] o ***Liferay*** [15].

Open Portal es una de las primeras herramientas que se pueden hallar al realizar una búsqueda de herramientas de licencia de código abierto que trabaje con *portlets*. Pero en este caso no es más que una quimera en comparación a otras herramientas o gestores de contenidos. La última versión data de 2001 y a pesar que como la mayoría de herramientas de licencia de código libre se nutre del desarrollo de la comunidad no parece haber recibido demasiado apoyo y su *release*, como su obsoleta página de presentación [13] ya remarca, es muy limitado y tiene por cubrir primeras necesidades en funcionalidades que otras herramientas ni mencionan y dan por sentadas.

En el caso de *uPortal* encontramos una herramienta mucho más perfilada y acabada. Entre otras cosas podemos ver que su última versión data de abril de 2012 y hay una comunidad que respalda el proyecto. Podemos encontrar fácilmente algunos videos [16] con demostraciones y ejemplos de lo que se puede hacer con la herramienta o de como configurarla.

Entre otras cosas *uPortal* ofrece varios tipos de descargas. Varias de ellas con la idea de dar un vistazo rápido a la herramienta. Esto se debe a que su objetivo es acercarse a un público más académico, lo cual no impiden que se pueda usar para el desarrollo [14].

Buscando herramientas más recientes y con mayor soporte nos cruzamos con nombres como *Drupal* [17], *OpenText* [18] y la ya mencionada *Liferay*. Entre estas la mejor valorada es *Liferay*.

Como ejemplo tenemos el informe '*Magic Quadrant for Horizontal Portals'* de *Gartner Group* [19]. Del que podemos encontrar varios *reviews*. El cuadro publicado en el último informe coloca a *Liferay* (una vez más) entre los líderes.

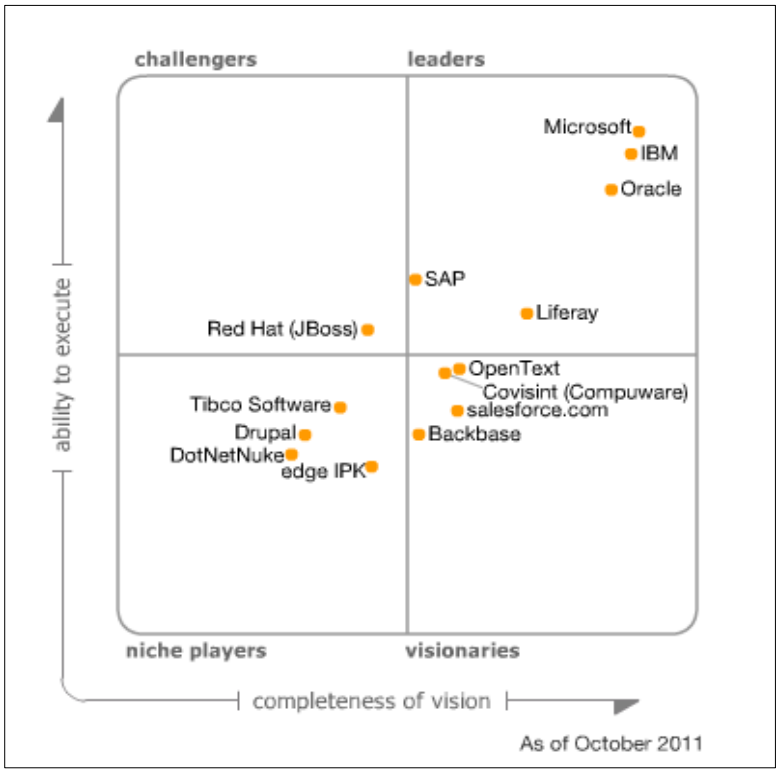

Figura 3.1 - Magic Quadrant for Horizontal Portals de Garnet inc. Publicado en Octubre de 2011

Según este informe, *Liferay* al estar basado en Java ofrece facilidades al desarrollo y se puede

comparar a varios productos propietarios. Además se destaca el trabajo con estándares abiertos y la fácil integración de herramientas de este tipo.

No menos importante destacan la forma de licencia de la herramienta. Esta nos proporciona dos tipos: *Community* y *Enterprise*. Siendo la primera gratuita y la segunda incluye soporte para, sobretodo, empresas profesionales del sector o que busquen una alternativa (algo más económica) a los productos más importantes, véase *Microsoft*, *IBM* o *SAP*.

En referencia a las dos otras herramientas el informe incluye a *Drupal* por primera vez y desplaza a *OpenText* de la zona de aspirantes a la de visionarios.

De *Drupal* se menciona como el mercado empieza a verlo (junto a otros de características similares) como opciones viables a pesar de no tener una visión de portal ajustada a las necesidades actuales. Aun así creen que hay posibilidades de que en no mucho tiempo se afiance en el mercado.

Por lo que se refiere a *OpenText* su reubicación en el cuadrante se debe precisamente a que la visión de portal se acerca más a las necesidades del mercado pero sigue estando por debajo de los líderes y tiene un problema con el soporte y algunas funcionalidades según los propios usuarios de la herramienta.

Quizás este sea el punto más interesante de *Liferay*, el soporte. En el caso de la versión gratuita lo da la comunidad y podemos encontrar gran cantidad de tutoriales y canales de comunicación para solventar muchos de los problemas que se puedan encontrar. Esto unido a la gran cantidad de opciones de descarga que ofrece son los puntos que más las diferencian y que ha permitido que avance tanto.

3.3 Por qué Liferay

Después de todo lo comentado y de un vistazo al abanico de herramientas se elige *Liferay* para el análisis de cara al desarrollo de portales para entidades deportivas.

De todas las herramientas consultadas está es la que mayor versatilidad ofrece. No solo por la cantidad de versiones que existen sino por la variedad de posibles configuraciones que encontramos. No solo cuenta con una versión *stand alone* para correr sobre cualquier servidor que lo soporte sino

que también disponemos de varias descargas en las que viene ligada (*bundled*) a un servidor (*Tomcat, Geronimo, Glassfish, Jboss, Jetty* y otros varios [20]).

En el caso de los *bundled* al ejecutarlos por primera vez advierte que tiene su propia base de datos pero que en ese momento puede especificarse otra si se desea para empezar a trabajar.

Como soporte añadido vemos que cuenta con diversas herramientas de cara a facilitar el desarrollo como *plug-ins* y extensiones para varios *IDEs*. Lo cual no es estrictamente necesario pero facilita enormemente el trabajo con *Liferay* y lo hace mucho más intuitivo.

4 Diseño

4.1 Casos de Uso

Dados los requerimientos hay que tener en cuenta que los usuarios del portal son todos aquellos que naveguen por este. Este usuario sin identificar no solo podrá navegar por la mayor parte del contenido público del portal sino que también podrá interactuar con este, apuntándose a eventos o comentando noticias. También puede inscribirse en el club, pero está acción repercutirá convirtiéndole en socio, obligando a pagar las cuotas y a esperar confirmación de la entidad.

En el caso de los usuarios registrados además podrán modificar sus datos (algunos, no todos).

El administrador por su lado podrá generar informes, gestionar altas y bajas de activos del club (socios, patrocinadores), crear y administrar eventos y noticias, acceder a algunas opciones de configuración del portal y comentar y moderar los comentarios.

Hay que tener en cuenta que los socios hasta que no se identifiquen serán usuarios sin registrar. La opción de identificación no tiene porqué ser inmediata tras el registro.

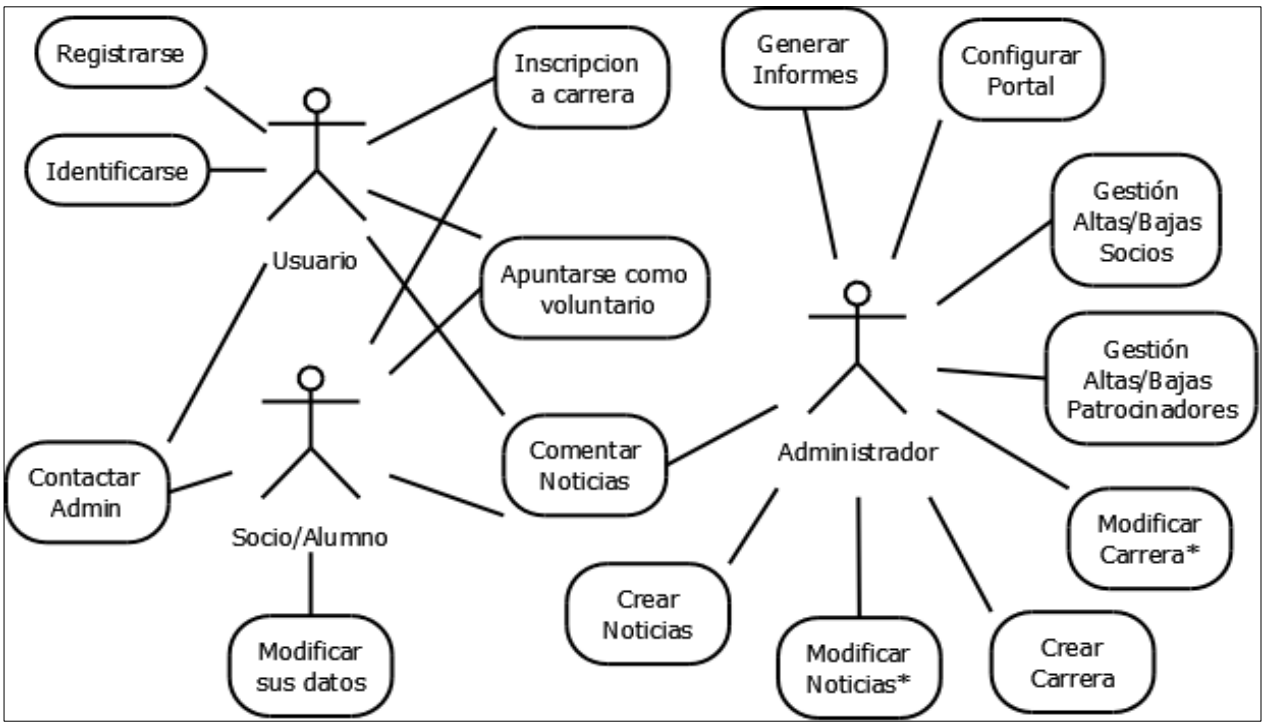

Figura 4.1 – Diagrama de casos de uso de los usuarios del portal.

4.2 Base de datos

Según los requisitos que se nos proporcionan se ha diseñado una sencilla base de datos que satisface las necesidades y exigencias del cliente. En este caso existen varias restricciones que se nos imponen y que dan lugar a varias tablas y relaciones que no se han contemplado con los requisitos recopilados en un capítulo anterior.

Cabe reseñar que la existencia de los eventos es lo que da mayor riqueza a la base de datos y también es el principal punto de debate en la fase de diseño.

Algunas condiciones pueden llevar a inconsistencias en los datos. Las condiciones polémicas son:

- No es necesario formar parte del club para participar en un evento.
- Se distinguen participantes en un evento de los voluntarios del mismo.
- Toda persona que participa de un evento dará sus datos.
- Se requiere guardar los datos de voluntarios y participantes.

Con esto en mente a la hora de diseñar la base de datos aparece una tabla que tras recoger los requisitos no se comentó, el corredor. Entendiendo al corredor como aquel que compite en un evento sin formar parte del club.

Tablas

Las cuatro tablas que contiene información sobre personas (socio, alumno, voluntario y corredor) comparten los siguientes campos entre ellas: nombre, primer apellido, segundo apellido, DNI, correo electrónico, fecha de nacimiento, dirección de residencia y teléfono. Estos serán los datos que deberán introducirse en la tabla de voluntario sin requerir nada más.

El corredor además deberá incluir el número de cuenta bancaria para facilitar el pago por la participación en el evento y tendrá la posibilidad de facilitar el número de chip de corredor.

En el caso del socio también incluye los dos datos del corredor, además de la posibilidad de almacenar hasta tres marcas personales de tiempos en carrera (el cliente no especifica sus

restricciones), foto, especialidad (entre las tres que distingue el club, pista, carretera y montaña), los datos de identificación en el portal (nombre de usuario y contraseña) y finalmente la talla de camiseta ya que el club proporciona uniformes a los corredores.

El caso del alumno incluirá además de los campos que comparten estas cuatro tablas: la cuenta bancaria, foto, talla, datos de conexión al portal, nombre y apellidos del tutor. Destacar que a pesar de que comparten datos no todos ellos se tratarán de la misma forma ya que según al tipo de persona que se refieran se utilizarán de forma distinta. De modo que las cuatro tablas hasta ahora quedan de esta forma:

Socio	Alumno	Voluntario	Corredor
Nombre	Nombre	Nombre	Nombre
Primer apellido	Primer apellido	Primer apellido	Primer apellido
Segundo apellido	Segundo apellido	Segundo apellido	Segundo apellido
NIF o equivalente	*NIF o equivalente*	NIF o equivalente	NIF o equivalente
Correo electrónico	*Correo electrónico*	Correo electrónico	Correo electrónico
Fecha de nacimiento	Fecha de nacimiento	Fecha de nacimiento	Fecha de nacimiento
Dirección	Dirección	*Dirección*	*Dirección*
Teléfono	Teléfono	Teléfono	Teléfono
Foto	*Foto*		Número de cuenta
Talla	*Talla*		*Número de chip*
Número de cuenta	Número de cuenta		
Número de chip	Nombre del tutor		
Nombre de usuario	Primer apellido del tutor		
Contraseña	Segundo apellido del tutor		
Especialidad	*Nombre de usuario*		
Marca 1	*Contraseña*		
Marca 2			
Marca 3			

Los campos que aparecen en cursiva son aquellos no obligatorios. Por ejemplo en el caso de los alumnos al ser menores de edad existe la posibilidad de que no tengan aún documento nacional de identidad así que el campo debe de poder ser nulo. No se plantea pero podría requerirse el NIF del tutor.

La opción de registrarse en el portal es básicamente una herramienta para gestionar altas y bajas en el club. Así pues no obliga a adquirir un nombre de usuario y una contraseña. Esto crea un posible problema de cara a participar en eventos que también repercutirá sobre el voluntariado de una forma similar y que se comenta más adelante.

Datos como la talla, el chip, la foto, las marcas personales o la especialidad no son datos estrictamente necesarios. Los datos de contacto y los identificativos de la persona si lo son. Para voluntarios o corredores no es obligatoria que proporcionen la dirección. Se necesitará también una tabla concreta para los eventos. También contamos con una tabla para las noticias y otra para los patrocinadores.

En un principio se prevé que los eventos sean por lo general carreras. A pesar de eso y en previsión de poder patrocinar también otro tipo de eventos (quedadas o conferencias) se permite que algunos campos puedan ser nulos. La tabla noticias tiene un formato similar al de las de un periódico. Pero da libertad a varios de los campos de forma que se podrán almacenar avisos (sin imagen o resumen). Para patrocinador existe un campo de actividad por una cuestión de altas y bajas.

Evento	Noticia	Patrocinador
Nombre	Titular	Nombre
Descripción	*Subtitular*	*Dirección sede*
Categoría	*Resumen*	Correo electrónico
Lugar	Contenido	*Logo*
Cuota	Categoría	Descripción
Distancia	*Foto*	*Comentarios*
Fecha	*Pie de foto*	Activo
Resumen	Fecha de publicación	
	Activa	

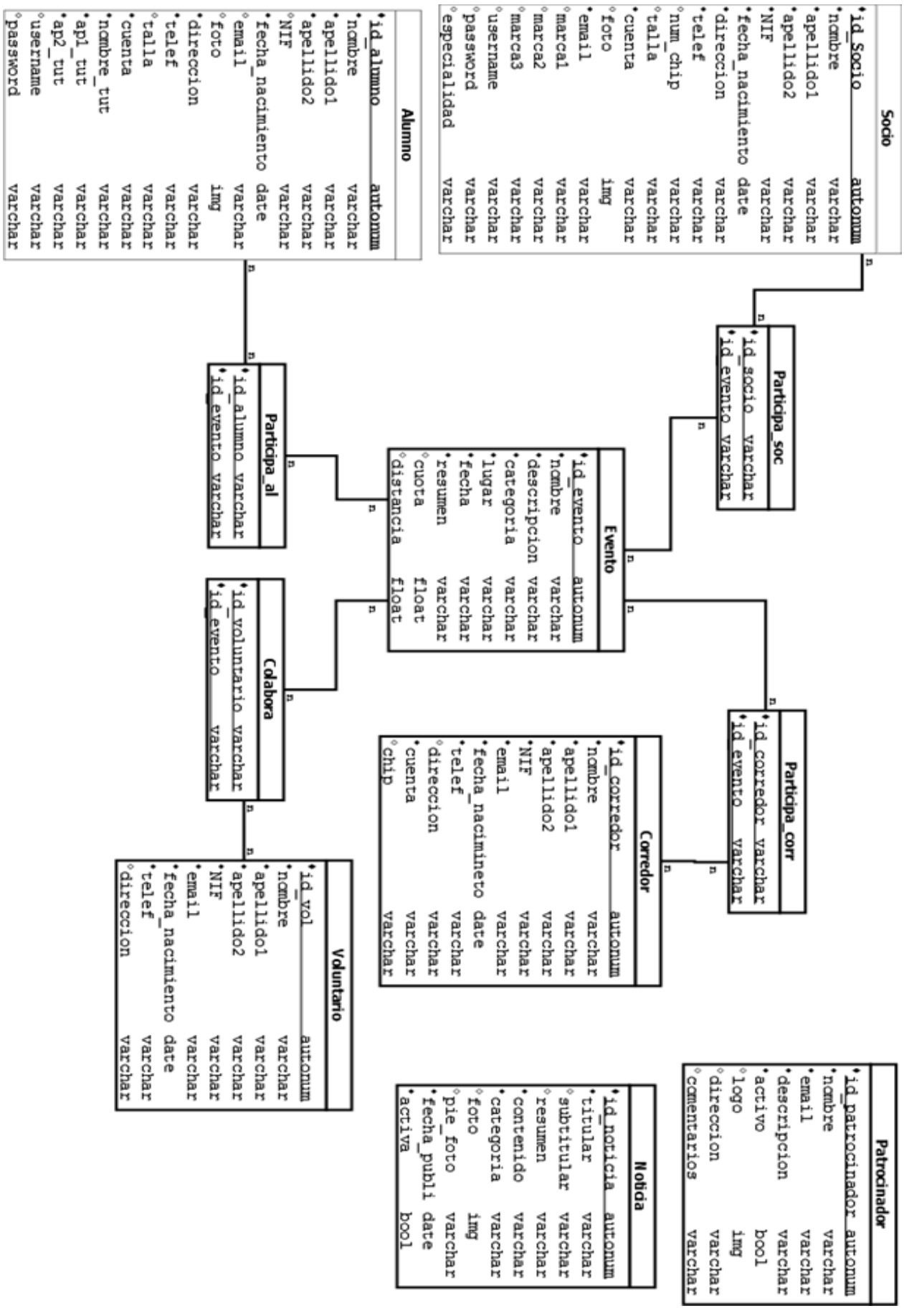

Figura 4.2 – Diagrama propuesto de la base de datos.

Las relaciones entre las tablas que representan personas y la tabla de eventos generan tablas intermedias. En caso de ser con corredores (ya sean alumnos, socios o corredores externos al club) se han llamado participa y en el caso de ser voluntarios colabora. Aunque en un principio no cuentan con datos extras se les podría añadir algún campo más como dorsal o categoría (si se hace distinción entre corredores según sexo, edad o modalidad) incluso se podría agregar un campo para determinar si el cobro de la cuota de inscripción al evento ya se ha realizado.

En la figura 4.2 faltaría añadir la tabla con los datos del administrador (usuario y contraseña) que permita identificarse y acceder al portal como tal.

Con este diseño ya aparecen unos posibles problemas dados por los requerimientos. Puede que en nuestra base de datos una misma persona aparezca en varias tablas. Por ejemplo, si alguien participa en varios eventos y no siempre como competidor.

Al no requerir un registro con identificación para acceder como participante en un evento hay que tener en cuenta que se deberá comprobar si los datos dados al inscribirse ya existen en la tabla pertinente. De esta forma campos como nombre, apellidos, fecha de nacimiento y NIF, que deben ser invariantes, tienen que coincidir al intentar volver a participar. Del mismo modo los demás datos si puede haber variado y se deberían actualizar.

La actualización de estos datos no tiene porque ser automática y es recomendable que sea validada por un integrante de la dirección del club.

Por otro lado como ya se ha comentado, tal y como el cliente especifica, en la base de datos se duplicarán las personas que hayan sido voluntarias y corredores. Tampoco se ha contemplado el hecho de que un socio del club quiera ser voluntario.

Esto se debe a que se ha requerido específicamente el mantener el voluntariado como una tabla específica. Así pues el socio no deberá volver a dar sus datos ya que al estar identificado se tiene acceso a estos, pero se le duplicará en la base de datos por especificaciones del cliente.

Teniendo en cuenta que la introducción del NIF es obligatoria y este campo es único entre personas se podría trabajar con esto para intentar solucionarlo pero sería preciso comprobar también otras coincidencias.

Posible restructuración

Una posible solución al problema que se nos presenta con la base de datos pasaría por cambiar algo la funcionalidad. Hasta ahora se ha priorizado el modelo que marcado por los requerimientos del cliente, pero obligando a registrarse a cualquier usuario que quiera interactuar con el portal y haciendo una diferenciación de tipos de usuarios entre los socios o alumnos (ambos podemos considerarlos usuarios de una versión *pro*) y los que no forman parte del club se puede solucionar el problema.

Con este cambio las tablas intermedias serían las que definirían si la participación en el evento es como corredor o voluntario. De todos modos esta solución choca con los deseos concretos del cliente y en el caso de hacerse el portal con fines de uso sería necesario discutirlo e intentar llegar a un acuerdo.

4.3 Estructura del portal

Una vez definidas las acciones y los datos pasamos a exponer como se estructuraría el portal. Tal y como se comentó en los requerimientos del cliente, el portal debería contar con las secciones del club, eventos (carreras), atletas, patrocinadores y un histórico de noticias.

La estructura es sencilla sin contar con secciones dentro de otras. Su navegación por lo tanto es sencilla y lineal.

Dado esto con una barra de navegación que de acceso a las distintas secciones tendremos el portal estructurado. Como podemos ver no haría falta crear un mapa web a no ser que se quieran añadir metadatos para facilitar su aparición en (esto podría ser de interés para los eventos o incluso para atletas y patrocinadores).

Según las indicaciones del cliente podemos sugerir una estructura para varias de la secciones. Por ejemplo en el caso del inicio una presentación que divida las noticias en cuatro mostrando la última (o últimas) de cada una de las cuatro secciones.

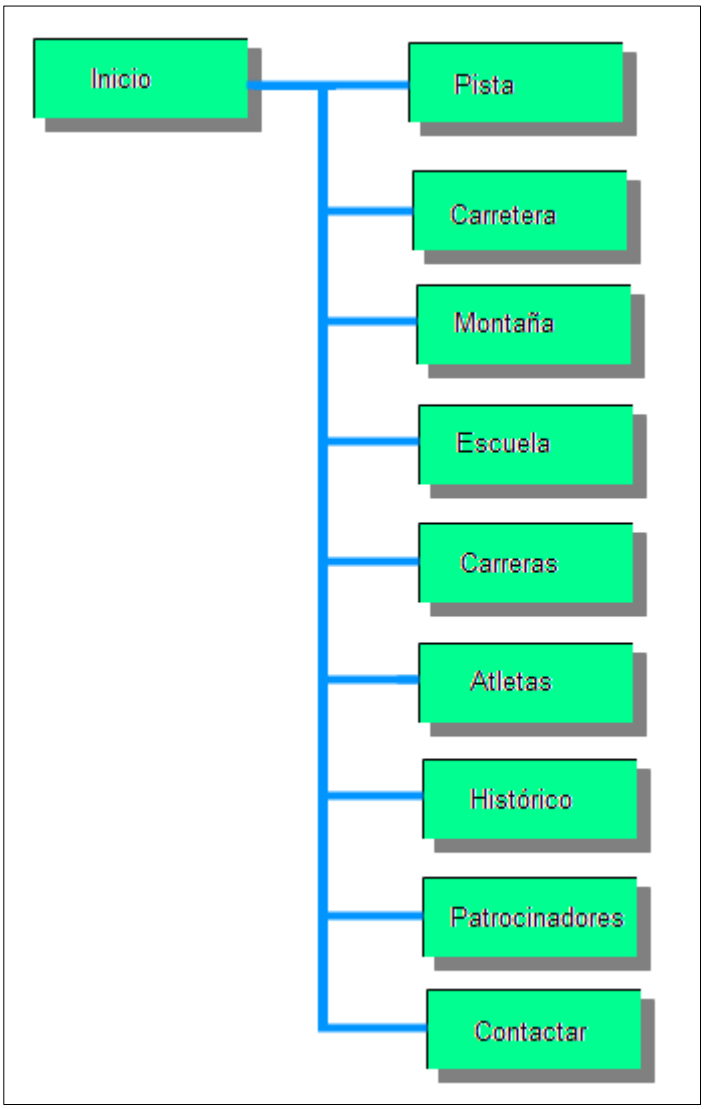

Figura 4.3 – Mapa web sugerido para el portal.

Con esto tenemos una presentación esquemática y elegante del contenido de la página de inicio. De todos modos también sería práctico añadir un espacio para anuncios de carácter general que quiera dar el club (aniversario de la fundación, felicitación de fiestas, enlaces a anuncios de conferencias u ofertas generales). Pudiendo añadir por encima de la subdivisión del inicio en cuatro, otra que aloje este espacio para comunicados sin clasificar o que no encaje en las secciones.

Las cuatro páginas del portal dedicadas a las secciones se pueden organizar como blogs donde aparecen las noticias o información de estás.

Por lo que se refiere a Atletas y Patrocinadores nos encontramos con un formato muy similar que presenta información de activos del club para todo aquel que quiera curiosearlos. A pesar del parecido seguramente sea más agradable estructurarlas de forma distinta, con un aspecto más comercial la página de patrocinadores (con algo más de separación entre cada uno de ellos´,

remarcando un espacio diferencial entre las marcas) mientras que el elenco de socios es más interesante buscar una presentación más listada y compacta dando una imagen de unidad y equipo.

En el caso de los eventos también parece interesante subdividirlos aunque más que por categorías, y con la previsión de un volumen no muy elevado de estos, se propone una vista que diferencia eventos activos de los que ya no los son. Entendiendo como activos aquellos que están siendo organizados o en previsión (si aparecen en la base de datos) e inactivos aquellos que ya han sucedido.

Del mismo modo que en el caso de las cuatro secciones del club, en el caso del archivo histórico una presentación estilo blog parece lo más interesante y una solución más que adecuada. En la sección de contacto la propuesta es una página similar a la de un perfil con datos interesantes sobre el club. Alguna imagen de las instalaciones, algo de historia (su fecha de fundación, intenciones, actividades habituales) y por supuesto los datos de contacto y su sede.

4.4 Clases, flujo de ejecución y validación

Clases

Dentro del funcionamiento del portal y para interactuar cómodamente con los datos que tenemos en la base de datos podemos crear las siguientes clases:

- Socio
- Alumno
- Voluntario
- Corredor
- Noticia
- Evento

Las clases Socio, Alumno, Voluntario y Corredor comparten múltiples atributos así que podemos definir una clase Usuario y hacer que las otras cuatro hereden de ella todo lo que tienen en común.

Las clases de Evento y Noticia no comparten demasiados atributos entre ellas ni con las referentes a personas. El contenido que tiene es el de las tablas de la base de datos con estos mismos nombres.

En la siguiente figura vemos un ejemplo con parte del código de la clase noticia.

```
public class Noticia {
    private int id;
    private String titular;
    private String subtitular;
    private String foto;
    private String texto;
    private String piefoto;
    private String fecha;
    private String resumen;

    public Noticia(){

    }

    public Noticia(int id ,String titular, String subt
        this.id = id;
        this.titular=titular;
        this.subtitular=subtitular;
        this.foto=foto;
        this.texto  = texto;
        this.piefoto = piefoto;
        this.fecha = fecha;
    }


    //Set & Get id
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }

    //Set & Get titular
    public String getTitular() {
```

Figura 4.4 – Fragmento de código de la clase Noticia.

Flujo de ejecución

Es interesante fijarse en el funcionamiento de los eventos. Esta es, quizás, la funcionalidad del portal más interesante y menos estándar entre todos los requerimientos de nuestro portal de ejemplo. Implicará inserciones en la base de datos tanto en su creación como cuando cumpla su función. En el caso de la creación responderá de una forma similar a como debe hacerlo la creación de noticias y en cierto modo la inscripción en la carrera se comporta como la inscripción de socios.

Es curioso el hecho de que a pesar de tener la clase Evento está solo aparece en los diseños de los diagramas de flujo en el de creación del evento. En cambio en el diagrama para la inscripción esta clase no interviene. De todos modos en el momento en el que mostramos el evento previamente se a instanciado un objeto de esta clase con la información procedentes de la base de datos para poder mostrarla de forma correcta.

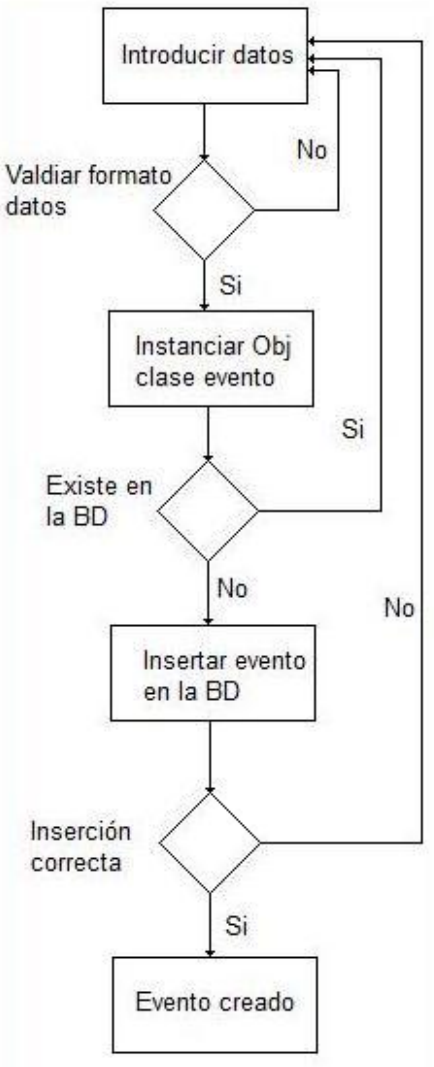

Figura 4.5 – Diagramas de flujo de ejecución de la creación de Eventos.

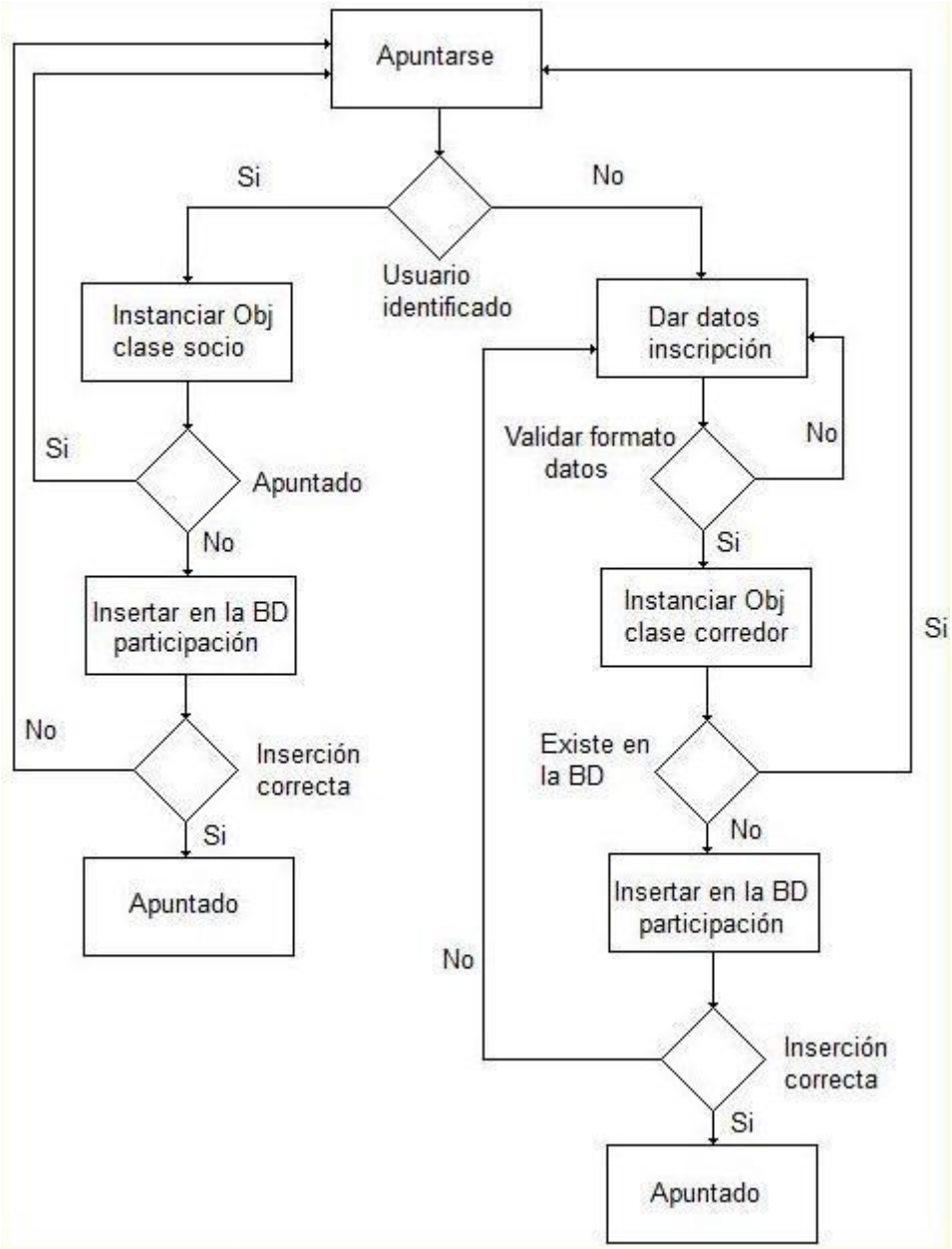

Figura 4.6 – Diagrama de flujo de la inscripción a Eventos.

La figura 4.5 muestra el funcionamiento de la creación de eventos y la 4.6 el de la inscripción en estos. Cada vez que volvemos a un paso anterior en una de las decisiones es conveniente añadir un mensaje de error. Habrá que distinguir entre un error de inserción en la base de datos de uno en el formato de los datos. El primer caso se recoge un error de la base de datos y el código *JAVA* será el encargado de gestionarlo. El segundo es un problema de validación y según a qué nivel se produzca se tratará de una forma u otra.

Validación

En general todo lo referente a la validación se puede gestionar con *JAVA*, pero en el caso de los

formatos de los datos en un formulario (por ejemplo el de inscripción de un usuario en el portal o el de creación de una noticia) también podemos hacerlo con *JavaScript*.

Inclusión del JavaScript en un formulario:

```
<script type="text/javascript"  src="altaUsu.js"></script>
```

Llamada a la función validar del JavaScript:

```
<input name="boton" type="button" value="Aceptar" onclick="validar()"/>
```

Muestra de código de la validación de datos de un formulario:

```
function validar(){
    var cadena = "El/los campo/s \n" + "\n";
    var cadfin = " son incorrectos y necesarios, por favor reviselos";
    var focus = false;
    var a=1;

    if(document.getElementById("usu").value.length<1){
        document.getElementById("usu").style.backgroundColor="#FF0000";
        cadena = cadena + "Usuario \n";
        if(!focus){
            document.getElementById("usu").focus();
            focus = true;
        }
    }
    else document.getElementById("usu").style.backgroundColor="#FFF";

    if(document.getElementById("pass").value.length<1){
        document.getElementById("pass").style.backgroundColor="#FF0000";
        cadena = cadena + "Palabra clave \n";
        if(!focus){
```

Figura 4.7 – Fragmentos de código de un JavaScript.

En la figura 4.7 se puede ver como en el *JavaScript* se crea una cadena en la que se van añadiendo los campos que no superan la validación para luego mostrarlos en un mensaje de error. El fragmento de código también muestra que este *script* cambia el color del fondo del campo del formulario a rojo para que además sea más visible el error para el usuario.

Por otro lado hay otros tipos de validación una vez superado el *JavaScript*, por ejemplo el hecho de intentar insertar datos que ya existen. Un ejemplo sería el de un usuario que intenta darse de alta con un NIF de otro socio o algún otro campo único. En estos casos hay que comprobar en la base de datos que no suceda antes de generar la consulta que añadirá el nuevo usuario.

```
try{

    //hacer comprobacion de si el dni ya existe...
    ResultSet cdr=BD.obtenerSQL("SELECT dni FROM socios WHERE cliente='" + u

    if(cdr.next()){
        error = "Hay un error en los datos.";
        chi = "/altacl.jsp";
    }
    else{
        int b=0;
        //generamos la consulta
        String consulin = "INSERT INTO socios (username,password,nombre,apel
```

Figura 4.8 – Comprobación de datos antes de una inserción en la Base de Datos.

En la figura 4.8 se aprecia el anterior ejemplo. Como es un acceso a la base de datos el código debe situarse dentro de un *try-catch* ya que es la forma adecuada de proceder y así se podrán generar mensajes de error en caso de que haya un fallo de acceso. Así pues la validación puede generar varios mensajes de error dependiendo de donde falle.

En el caso de que el fallo sea por duplicidad en la base de datos no se debe proceder mostrando que datos son los ya existentes ya que se puede aprovechar esto para conseguir (o comprobar) información confidencial.

5 Análisis

5.1 Toma de contacto con Liferay

Arquitectura

Liferay está basado en *JAVA* así pues corre sobre un sistema operativo que tenga instalada una *JVM*. Del mismo modo el servidor que habrá por encima de la *JVM* proporciona servicios como *Java Mail*, *JMS* o *JSP*/*Servlet*. En la siguiente capa tenemos ya el grueso o núcleo de *Liferay*, dando cabida a los túneles que se usarán para los *portlets* creados por el usuario, tecnologías y *frameworks* como *Hivernate*, *Spirng* o *JBP* y adaptadores de distintos lenguajes (como *RUBY* o *PHP*) facilitando su integración [34].

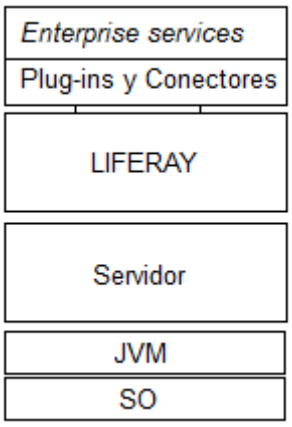

Figura 5.1 – Arquitectura lógica de Liferay

A más alto nivel existe una capa llamada *Enterprise services* y ofrece soluciones sobre todo de gestión de documentación, contenidos (web y empresariales), seguridad y flujo de trabajo proporcionando herramientas de colaboración, *virtualización* o algunas opciones más de *tuneling*. Justo por debajo encontramos los *plug-ins* de los distintos módulos de *Liferay* (comentados más adelante), el núcleo de administración y algunos conectores con los servicios de la capa superior.

Versiones

Tras elegir la herramienta sobre la que trabajaríamos, *Liferay*, hubo que decidirse por la versión y el formato. En el caso de la versión se optó por la última de forma natural, ya que siempre son las más

completas. El último *release* de una versión final es bastante reciente (principios de 2012).

Como ya se ha comentado el producto en su versión *Community* viene con varios paquetes para elegir. Estos básicamente ofrecen un servidor con el gestor de contenidos ya instalado (y un *site* de demostración) y algunos packs más útiles (o necesarios) para trabajar [20].

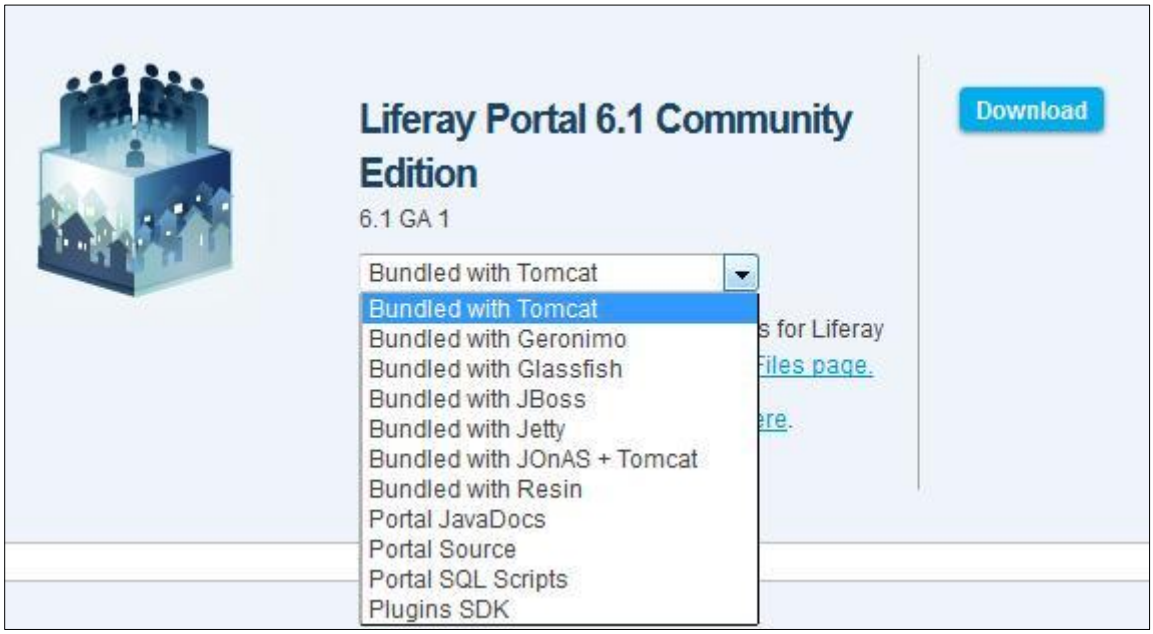

Figura 5.2 – Descargas de la versión 6.1 de la Community Edition de Liferay.

Al no tener ninguna restricción ni por parte del cliente ni por guión y tras sondear por la red algunos comentarios de los usuarios sobre la nueva versión [21][22] se optó por utilizar el *bundled* con *Tomcat* (en algunos casos como con *Glassfish* había comentarios de que aun no era estable).

Instalación

Como se ha trabajado con el paquete incluido en *Tomcat* la instalación de *Liferay* en sí misma no ha existido. Simplemente al arrancando el servidor este ya carga todo lo necesario y lanza el portal de entrada desde el que se puede empezar a trabajar. La primera, el servidor pedirá que base de datos se quiere usar (la que trae por defecto o una propia).

Después de esto se puede empezar a ver algunas de las funcionalidades y lo que puede llegar a hacerse con la herramienta con el portal de demostración. Este es accesible desde la página de entrada a no ser que se eliminen los componentes que dan acceso o se borren los archivos de demostración de la instalación [23].

Para poder realmente trabajar y desarrollar es necesario descargar e instalar el *SDK* de la herramienta [24]. Esta es poco intuitiva y es recomendable el uso del algún *IDE*. Si se busca, en el mismo portal de la herramienta hay documentación varia para preparar *IDEs* de cara a desarrollar con *Liferay* [25].

Una vez más y de la misma forma que se procedió con la elección del paquete de descarga se consulto documentación y comentarios. Barajando varias opciones la que cuenta con mayor soporte es *Eclipse*. Siendo una de las herramientas más comunes en el ámbito de programación *JAVA* cuenta con su propio *plug-in* para el SDK de *Liferay*.

En la misma wiki de la comunidad encontramos una guía para la instalación en *Eclipse* de su *IDE* particular. La instalación de este *plug-in* requiere de:

- *Java 5.0 JRE*.
- *Eclipse Inidigo (3.7.x)* o *Eclipse Helios (3.6.x) IDE for Java EE Developers*.

Eclipse cuenta con herramientas para descargar e instalar Software nuevo. Utilizando la búsqueda *'Liferay IDE'* en la herramienta de eclipse y con la *url* que nos proporciona la guía, en apenas once pasos (muy básicos, el resumen podría hacerse en cuatro pasos) tendremos el *plug-in* disponible.

Una vez hecho esto habrá que configurar el *plug-in* del *SDK*. Este, aparte de lo requerido en la anterior instalación, también necesita tener instalado *Liferay Plug-ins SDK 6.0* o superior. En la wiki de la comunidad también hay documentación al respecto y además incluye instrucciones para probar el portal desde *Eclipse* con instrucciones de la instanciación del servidor y la creación del primer *Plug-in project*. Con esto ya está el entorno de trabajo preparado para trabajar en *Eclipse*.

5.2 Modulos de Liferay

Antes de entrar en los módulos cabe remarcar que *Liferay* utiliza diversas tecnologías dentro del campo del desarrollo de portales web. Estas son principalmente: *JAVA*, *JSP*, *XML*, *CSS*, *HTML*, *Javascript-Jquery* y *Velocity*.

Podemos separar las herramientas que ofrece *Liferay* para su análisis en cuatro bloques

diferenciados: *portlets*, *hooks*, *layouts* y temas. Los últimos dos se compaginan entre ellos y se pueden analizar de forma conjunta.

Portlets

Los *portlets* [12] se pueden ver como sencillas aplicaciones autocontenidas regidas por estándares (JSR 168 y JSR 286). Esto hace que cualquier *portlet* pueda adaptarse para cualquier portal que soporte sus estándares.

Un *portlet* empieza por una clase java que implementa la clase interfaz 'javax.portlet.Portlet'. Los *portlets* solo tienen sentido cuando están contenidos dentro de un receptáculo que suele ser el portal e interactúan con los usuarios a través de peticiones y respuestas.

Un *portlet* genera contenido dinámico para un portal y están desarrollados en *JAVA* cosa que los asemeja a los *servlets* [11]. Pero a diferencia de estos, los *portlet* no pueden ser llamados de forma directa, es decir, de un *servlet* se puede escribir su dirección web pero de un *portlet* solo se podrá obtener la dirección web de la página que lo contiene, pero no una petición directa sobre este. Otra diferencia es que los *servlets* pueden generar páginas o documentos completos, mientras los *portlets* solo generan una parte del contenido.

Podemos ver su funcionamiento y su arquitectura lógica de la siguiente forma:

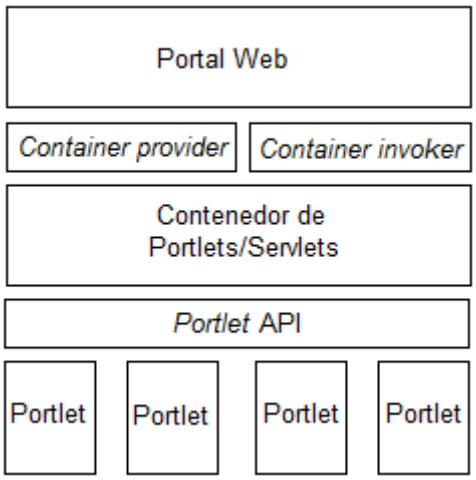

Figura 5.3 – Arquitectura lógica de un portlet.

Cuando hay una petición desde el portal, este hace una llamada al contenedor de *portlets* para cada

portlet esperando recibir su contenido a través del *Container Invoker*. A su vez el contenedor de *portlets* extenderá la llamada a los *portlets* mediante la *Portlet API*. El contenedor finalmente devolverá la información de los *portlets* gracias al *Container provider*.

Todas las funcionalidades y contenidos que nos ofrece *Liferay* son *portlets*. La idea es que además de estas podamos hacer nuestros propios *portlets* para realizar las tareas a las que no se les da solución con los que vienen integrados. El principal obstáculo es el hecho de que los *portlets* integrados en el núcleo de la herramienta son difíciles de modificar y si la funcionalidad que nos ofrece no se ajusta del todo a nuestras necesidades la mejor solución es desarrollar un nuevo *portlet*.

Hooks

Los *Hooks* [29] [34] son una herramienta propia de *Liferay* (en contraste con los *portlets* que están estandarizados). Estos están pensados como ayuda para modificar comportamientos del portal sin tener la necesidad de recompilarlo entero.

Básicamente, un *hook* nos permites:
- Sobrescribir *JSPs* del portal
- Monitorear la modificación o creación de las entidades de *Liferay*.
- Cambiar mensajes puntuales de la aplicación (‘*Language.properties’*).
- Modificar propiedades del ‘*portal.properties’*.
- Cambiar la configuración de forma dinámica.
- Poder ejecutar acciones ante determinados eventos del portal y de los usuarios.

Además en el momento en el que aplicamos el *Hook* veremos el resultado de forma inmediata y si decidimos retirarlo volveremos a tener el mismo funcionamiento que antes de aplicarlo.

La capacidad de modificar prácticamente cualquier parte del portal hace que estos abarquen una gran cantidad de lenguajes como *JAVA*, *JSP*, *HTML* o *XML*.

Los *Hooks* no permiten modificar cualquier cosa. El núcleo de *Liferay* no puede ser accedido con estos. Aun así no es imposible acceder al núcleo, existen los llamados *Ext Plug-ins* [30] [34].

Los *Ext plug-ins* proporcionan la forma más potente de aumentar las opciones que ofrece *Liferay* pero a costo de una complejidad muy elevada. Estos están pensados para utilizarse solo en casos

muy especiales en que ninguna otra herramienta permita llevar a cabo el trabajo deseado. Además estos trabajan de una manera tan específica que si cambiamos de versión de *Liferay* es muy probable que deje de funcionar correctamente el cambio hecho con *Ext plug-ins*. Además necesitan que se vuelva a reiniciar el servidor para comprobar su uso.

Como ejemplo de para que podría utilizarse un *Hook* está el poder añadir traducciones a algunos términos si manejamos diversos idiomas. Otro caso más acorde con nuestro ejemplo sería en el momento de registrarse como nuevo usuario en el portal del club, el hecho de poder sobrescribir un formulario *JSP* en el caso de que no fuera un socio sino un alumno y tuviera la necesidad de introducir otros datos. Este no es más que un ejemplo ya que este problema también se puede solucionar con campos restringidos según el marcado de una casilla en el formulario.

Layouts y Temas

La parte de visualización queda dividida en dos: *layout* y temas [31][32]. Así pues hay que planificarlos conjuntamente.

El *layout* se encarga de la disposición de los *portlet* dentro de la página, organizándolos en filas y/o columnas según la combinación que se diseñe. Cada página puede tener un único *layout*, pero varias páginas pueden utilizar el mismo. Una vez escogido el *layout* se podrán introducir los *portlets* distribuidos en las filas y columnas delimitadas por el *layout* [34].

Como ya se ha mencionado es importante planificar los *layout* y el tema de forma conjunta puesto que la estructura diseñada en *layout* será maquetada por el tema. De esta forma un *layout* puede ser reutilizado por varios temas y un mismo tema puede emplearse en varios *layout*.

Los *layout* están desarrollados con *HTML* y *Velocity*.

El tema se encarga de gestionar la apariencia del portal. Se desarrolla mayoritariamente en tres tecnologías: *CSS, Velocity* y *Javascript-Jquery*.

Es recomendable que se desarrolle a partir de un tema ya existente o de las bases que ofrece la *SDK* al generar un módulo de este tipo. Esto viene dado por la facilidad de pérdida de cambios al implementar tres tecnologías a la vez. Dentro del módulo tema, los ficheros se encuentran organizados en carpetas, agrupados según la tecnología que contienen.

Cuando se crea y se despliega un tema este aparece en el menú *manage > page >Look and feel*, en la parte inferior junto a los otros temas inactivos.

Podemos comprobar el cambio al crear un tema propio comparado con el look habitual:

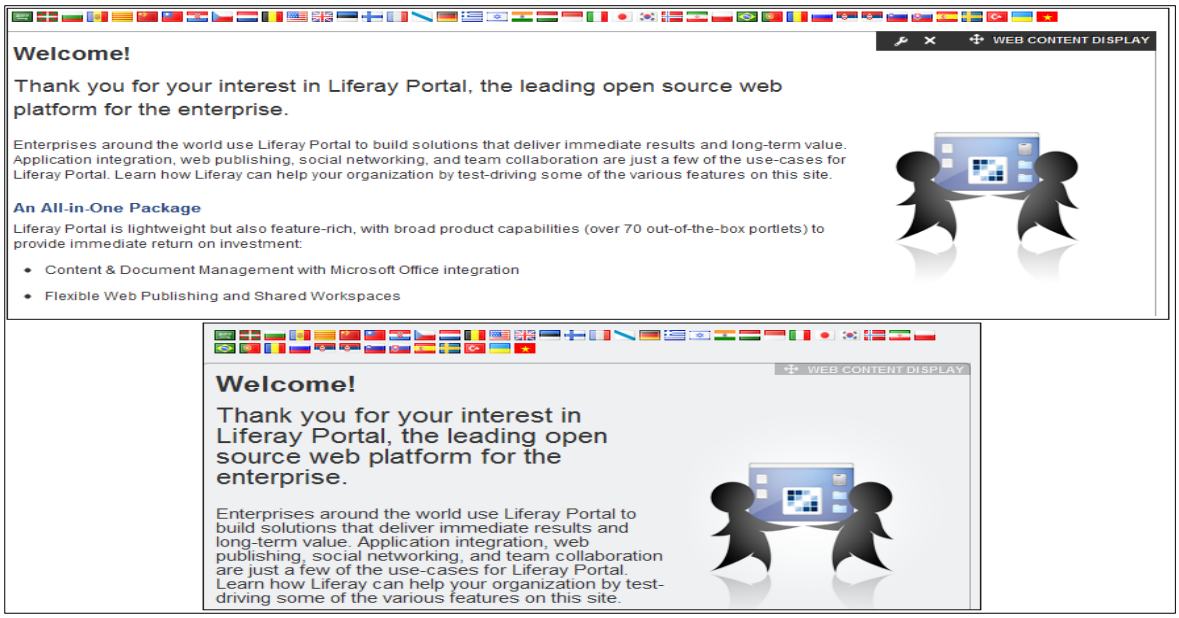

Figura 5.4 – Comparación de dos temas, el original y uno propio.

En la figura 5.5 se ve a la izquierda la separación por directorios. Nuestras modificaciones aparecen en el directorio *_diff*. Así pues cuando queramos modificar algún componente ya existente deberemos copiarlo junto a su estructura de directorio a la carpeta *_diff* y modificarlo en su nueva ubicación. A la derecha vemos los cambios realizados en la configuración (*xml*).

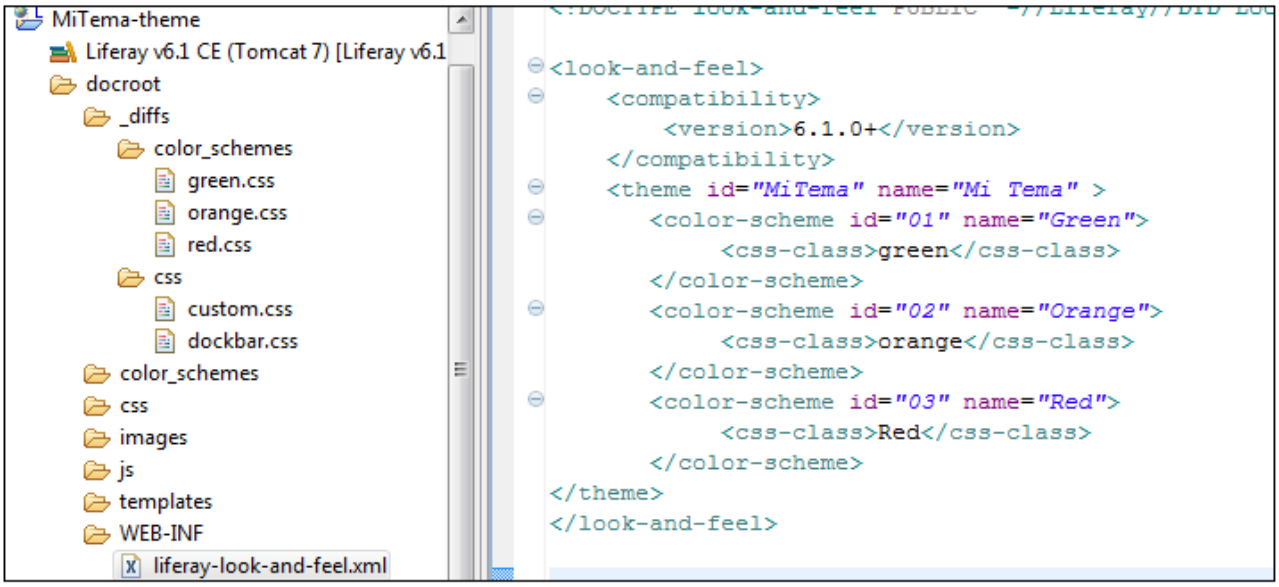

Figura 5.5 – Estructura de directorios de un Tema y código.

5.3 Trabajando con Portlets

Una de las principales cosas a tener en cuenta al trabajar con *portlets* es que estos lo hacen en dos fases. Esta es una diferencia de comportamiento que tienen con los *servlets* u otros entornos de trabajo. La necesidad de trabajar en dos fases viene dada por el hecho de que un *portlet* debe compartir su contendor (por ejemplo una página *HTML*) con otros *portlets*.

El portal es el encargado de generar la página llamando a todos los *portlets* y generando el código extra que requiera. De esta forma cuando el usuario interactúe con un *portlet* (por ejemplo clicando en un botón) lo hará solo con esa porción de la página. Esto hará que el portal actúe en consecuencia y al volver a hacer el *render* de la página deberá mostrar los cambios que puedan haber ocurrido en el *portlet* con el que a interactuado el usuario mientras el resto de *portlets* de la página repetirán la última invocación que se hizo de ellos [34].

Este comportamiento son las dos fases. El *portlet* con el que interactúa el usuario habrá pasado por la *Action Phase* y luego todos pasan por la *Render Phase*.

- Action Phase: Solo un *portlet* de la página podrá estar en esta fase a la vez, normalmente se producirá con la interacción del usuario y es donde se permite el cambio de estado del *portlet*. Una buena práctica es que si hay que modificar la base de datos desde un *portlet* se haga desde esta fase.
- Render Phase: Esta fase se lleva a cavo siempre (aunque no únicamente) después de la *Action Phase* y es para todos los *portlets* de la página. No hay especificaciones de en qué orden se ejecutarán cada uno.

Es necesario pasar información de una fase a otra del *portlet*. Para ello se dispone de los siguientes métodos:

Action Phase:

```
actionResponse.setRenderParameter("parameter-name", "value");
```

Render Phase:

```
renderRequest.getParameter("parameter-name");
```

Un ejemplo de esta utilización podría ser el siguiente (figura 5.6), en el que se aprecia cómo se obtiene el valor entrado por la acción (en este caso con un campo de formulario y enviado por un botón), se comprueba que no sea nulo y se almacena. En este caso el *portlet* (demo-portlet) nos permite generar un aviso y cambiarlo cuando se quiera.

```
<%@ page import="javax.portlet.PortletPreferences" %>
<portlet:defineObjects />
<%
PortletPreferences prefs = renderRequest.getPreferences();
String g = renderRequest.getParameter("g");
if (g != null) {
prefs.setValue("g", g);
prefs.store();
%>
    <p>Almacenamiento finalizado.</p>
<%
}
%>
```

Figura 5.6 – Código de ejemplo de un render Request.

La mínima expresión de un *portlet* debe contener tres componentes:

- Código *JAVA*.
- Fichero de configuración.
- Ficheros del lado del usuario (JSP, CSS, etc.).

Por ejemplo en demo-portlet (figura 5.7) vemos como contiene dos *JSPs*, que representan las dos fases con dos vistas distintas. Presenta un formato similar al de un pequeño proyecto.

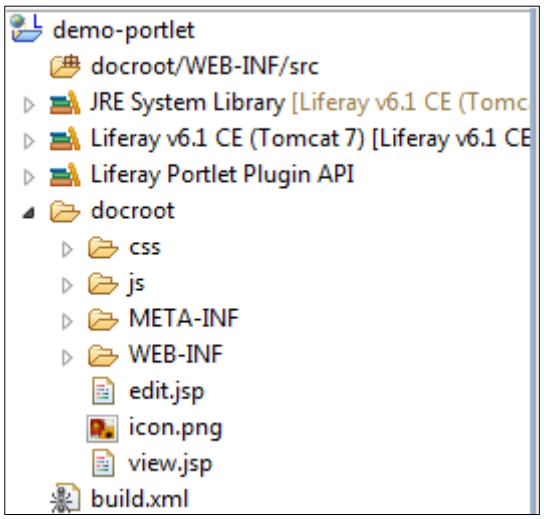

Figura 5.7 – Estructura de un portlet.

Si echamos un vistazo al fichero de configuración de un *portlet* (portlet.xml) podemos ver los siguientes campos:

portlet-name	Dentro de la aplicación es el identificador del *portlet*, tiene que ser único.
display-name	El nombre que se mostrara en las herramientas para acceder al *portlet*, no tiene porqué ser único.
portlet-class	Contiene el nombre de la clase que se encargará de las invocaciones al *portlet*.
init-param	Contiene una pareja de nombre/valor como valores de inicialización del *portlet*.
expiration-cache	Indica el tiempo que transcurre hasta que la respuesta (salida) del *portlet* expira. Un -1 indicará que nunca expira.
supports	Indica los tipos que soporta el *portlet* (por ejemplo HTML).
portlet-info	Autoexplicativo, información sobre el *portlet*.
security-role-ref	Especifica que roles pueden acceder al *portlet*.

Existen otros ficheros configuración de un *portlet* (como liferay-portlet.xml) que añaden algunos campos más de interés como podría ser el campo *instanceable* que indica si puede aparecer varias instancias del *portlet* en la misma página.

Todos estos archivos de configuración aparecen al crear un *portlet*. Pero, ¿Cómo se crea?

En el caso de estar utilizando el *SDK* [33] disponemos de un archivo (en el caso de trabajar en Unix create.sh y en Windows create.bat) que al ejecutarlo con los parámetros adecuados (nombre del proyecto, no puede contener espacios, y nombre de visualización) nos creara el *portlet* (uno vacio sobre el que trabajar).

Del mismo modo una vez hayamos desarrollado nuestro *portlet* deberemos desplegarlo. Para hacer esto *Liferay* proporciona un mecanismo llamado *autodeploy*. De un modo similar a la creación solo debemos situarnos con la consola en el directorio de nuestro *portlet* y ejecutar *'ant deploy'* y esperar a recibir el mensaje de que el *portlet* ha sido desplegado con éxito.

En el caso de utilizar Eclipse [25] esto se hará a trabes de los menús. Para crear un *portlet* debemos seguir estos pasos, ir a *File > New Project... > Liferay > Liferay Plug-in Project* y nos aparecerá un asistente para terminar su creación. Desde aquí podemos ponerle nombre (como los parámetros del

SDK) y seleccionar *portlet* (también se pueden crear *Hooks, Ext Plug-ins, Layouts y Themes*).

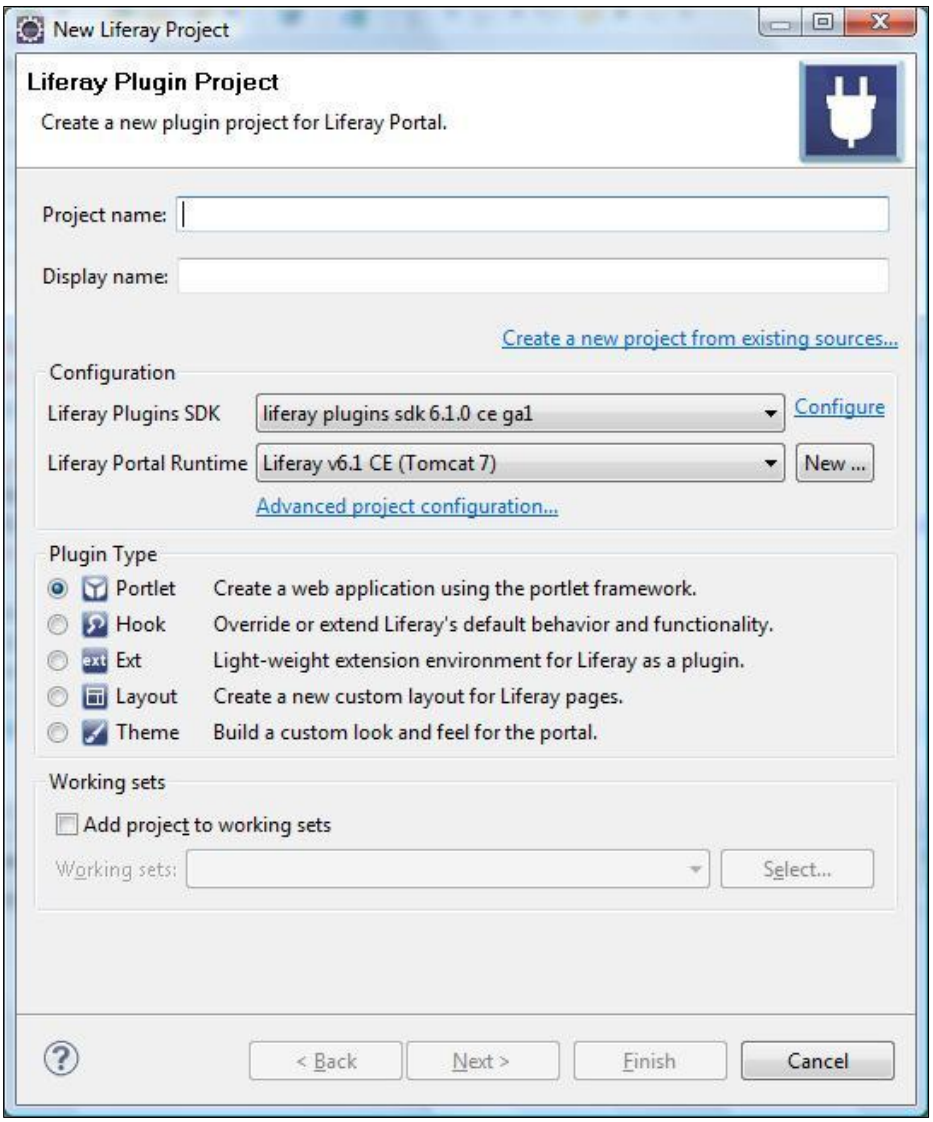

Figura 5.8 – Asistente de Eclipse para crear proyectos Liferay.

Una vez desarrollado el *portlet* al igual que con el SDK deberemos desplegarlo. Para ello iremos a la pestaña de servidores de Eclipse y desplegando el menú con el botón derecho del ratón sobre el servidor en el que trabajamos aparecerá la opción *'Add and remove...'* que nos permitirá desplegar o recoger nuestros *portlets* u otras herramientas.

Podemos incluir clases *JAVA* y páginas *JSP* para crear nuestras funcionalidades (en principio usaremos una *JSP* distinta para cada vista que queramos añadir al *portlet*). Podemos incluir archivos de visualización *css* y *javascripts*.

En la figura 5.9 podemos ver que el *SDK* nos genera todo el sistema de ficheros y carpetas y añade las librerías necesarias para su funcionamiento.

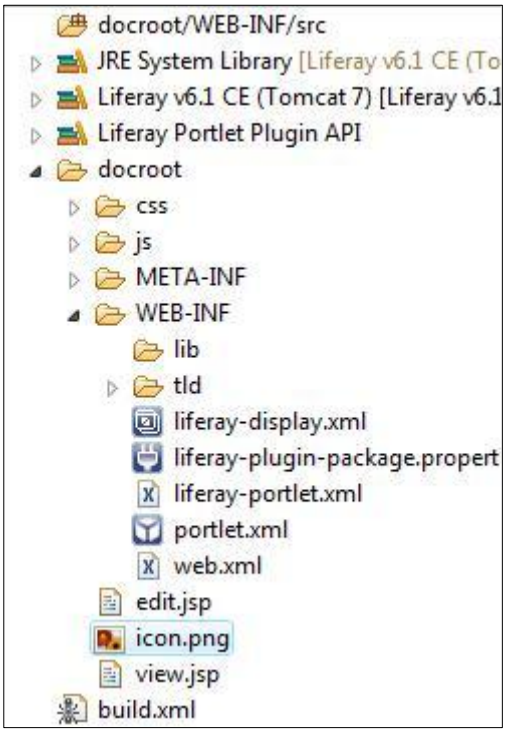

Figura 5.9 – Estructura de ficheros de un Portlet.

5.4 Ubicando Portlets y configurando la vista del portal desde Liferay

Desde el portal que se abre al ejecutar el servidor (o en el caso de estar el servidor ya corriendo en la página de inicio) podremos ya empezar a gestionar nuestro portal [34]. Contamos con un menú en la parte superior del portal que nos permitirá ver las distintas opciones para trabajar.

Figura 5.10 – Barra de herramientas de trabajo en Liferay.

Desde la pestaña de 'Add' se nos abrirá un desplegable con múltiples opciones la última de las cuales es 'More...'. Esta nos abrirá un pequeño cuadro desde el que podemos acceder a las distintas aplicaciones (*portlets*). Aparecen subdivididos por categorías y el mismo cuadro permite una búsqueda. En el caso de los *portlet* que hayamos desplegado nosotros aparecerán en la categoría *'Sample'*. En la figura 5.11 aparece el *portlet* Demo-portlet del que sea comentado la estructura de ficheros y que ya ha sido añadido (y al no estar habilitado su campo *instance* no podrá repetirse en esta página del portal).

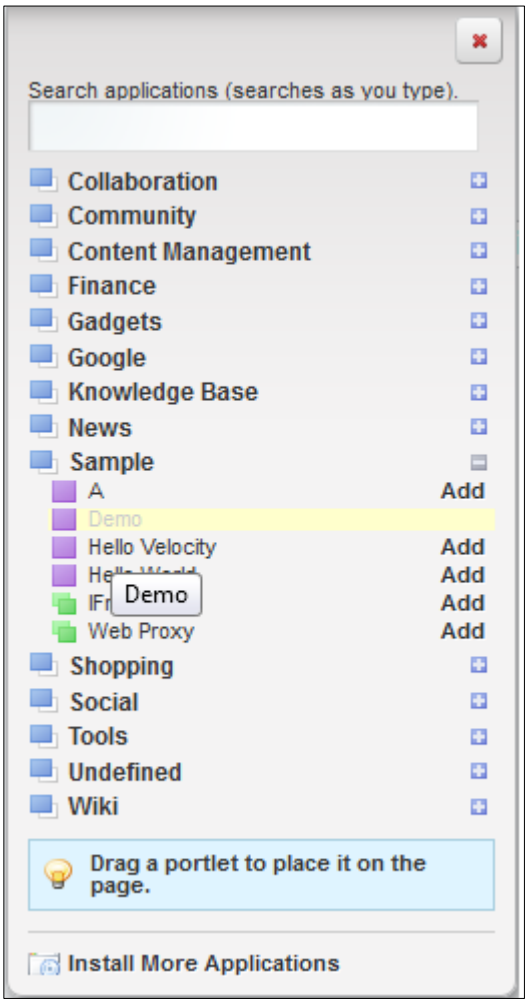

Figura 5.11 – Desplegable que contiene los portlets a disposición de Liferay.

Una vez elegido el *portlet* que se quiere añadir solo es necesario arrastrarlo desde el cuadro a la página para colocarlo. El *portlet* será operativo de inmediato y se podrá editar y eliminar desde el mismo *portlet* si se cuenta con los permisos necesarios.

Para organizar el resto de elementos del portal y otros elementos como podrían ser los *Layouts* se cuenta con el menú de la segunda pestaña. Es de destacar la opción de *'Site Membership'* que permite gestionar una de las funcionalidades más interesantes de *Liferay*, usuarios (comentado a continuación).

5.5 Sistema de gestión de usuarios

Liferay proporciona una gran cantidad de opciones de cara a permisos y organización de usuarios. Esto le otorga una versatilidad importante pero va acompañado de una complejidad elevada. En el caso del club que sirve de cliente no habría gran diferencia entre los permisos de la mayoría de usuarios del portal. Pero de cara a una abstracción más elevada es interesante tener esta prestación y

conlleva el tener bien diferenciados los usuarios.

Contamos con la existencia de *organizaciones*, *comunidades* y *grupos de usuarios*. Las organizaciones tienen una estructura jerárquica donde se pueden definir dependencias padre/hijo. Las comunidades tienen su razón de ser en las motivaciones e intereses de los usuarios, pudiéndolos clasificar según estos. Por último, los grupos de usuarios nos sirven para poder administrarlos de forma colectiva sin necesidad de hacerlo uno por uno.

Con el mismo fin contamos también con la existencia de roles. Los roles juegan un papel fundamental en la estructura y ayudan a diferenciar distintos usuarios dentro de una misma agrupación de las anteriormente nombradas.

Dentro de cada organización o comunidad existen una serie de roles que determinan los permisos del usuario. Por defecto existen cuatro roles diferentes: *Administrator*, *Owner*, *Content Reviewer* y *Member*.

Estos roles tienen una jerarquía y pueden llevar a cabo distintas tareas y manipular el contenido de la agrupación. En la siguiente tabla vemos más o menos que repercusión tiene cada uno:

Rol	Tarea	Permisos
Owner	Administrar la agrupación.	Generar nuevos administradores.
Administrador	Gestionar la agrupación.	Añade members y content reviewers.
Content Reviwer	Gestionar contenido y añadir *portlets*.	Añadir determinados *portlets*.
Member	Ninguna	Visualiza páginas privadas.

Este entramado de permisos y agrupaciones permite gestionar gran cantidad de usuarios y diferenciarlos y es muy potente y atractivo para grandes organizaciones, sean o no deportivas. Para gestionarlo se puede hacer desde la opción '*Manage*' > '*Site Memnership*'.

5.6 Inicio

Tal y como se plantea la página de inicio del portal, necesita una estructura muy concreta. La idea es mostrar cuatro porciones distribuidas en dos columnas (dos y dos) con las noticias más recientes de las cuatro secciones del club. Aun y así antes de esta subdivisión debe quedar por encima un espacio para los avisos.

La barra de navegación viene dada por el contenedor que aporta *Liferay* para *portlets*. Para organizar el resto de la página necesitaremos un *layout* que subdivida en dos columnas el contenido y que además tenga un espacio por encima para poder añadir un *portlet* que ocupe todo el ancho de la presentación.

Con esta presentación tenemos ya la estructura del inicio. El contenido queda cubierto por cinco *portlets*. En un principio de dos tipos, aviso y resumen de noticia.

Aviso

Este *portlet* es un sencillo mecanismo de comunicación editable por el administrador (por regla general un usuario con permisos). En su fase de activación permite modificar el mensaje que muestra y en la fase de *render* solo lo muestra. Esto se plantea así teniendo en cuenta que estos avisos no forman parte de la estructura de la base de datos y que por defecto tendremos un saludo a los usuarios (*greeting*).

En caso de que se quiera guardar esta información en la base de datos podemos tratarla como noticias, con la misma estructura de estas y catalogarlas como avisos en su categoría. En ese caso el *portlet* sería como el de resumen de noticia.

Resumen de noticia

Para este caso debemos tener en cuenta que la misma página contendrá diversas instancias del mismo *portlet* así pues en el *XML* (liferay-portlet.xml) correspondiente debemos configurar que sea instanciable.

El *portlet* en cuestión debe acceder a la base de datos y dependiendo de qué sección de noticias represente, obtener la última (o las últimas si se muestran varias) noticia de la categoría en cuestión.

Con una consulta a la base de datos es suficiente. El procesamiento de estos datos es bastante sencillo. Teniendo en cuenta que contamos con la clase Noticia, que nos permite crear objetos con los distintos atributos de está, podemos almacenar esta información para tratarla. Tratándose solo de la noticia actualizada en la página de inicio podemos acotar la consulta y obtener solo los campos

que nos interesan (por ejemplo titular, subtitular, resumen y fotografía) y manipularlos sin necesidad de crear un objeto de clase noticia.

5.7 Secciones del club y noticias en el histórico

Estos casos (Carretera, Pista, Montaña, Escuela e Histórico) son similares al de la página de inicio en cuanto a contenido. Contando con la diferencia que querremos mostrar el contenido completo de la noticia y todas las noticias no solo la última. De todos modos esto último es relativo ya que llegada cierta cantidad su visualización sería poco elegante y nada práctico.

Por esto aunque la consulta será de todas las noticias solo se muestra por pantalla cierto número (en principio las más recientes). Hay que añadir pues la lógica para navegar por las noticias subsiguientes mediante botones (por ejemplo) que cambiarán el estado de nuestro *portlet* (por lo tanto será en la fase de acción).

Así pues en la fase de *render* tenemos una consulta y desde la fase de acción recibiremos parámetros indicando donde empezar a mostrar los resultados de esta. Por defecto este debe ser un cero (o como se decida indicar que no avanzamos).

Por lo que respecta a la visualización solo contaremos con un *portlet* que nos mostrará todas las noticias existentes. Más allá de la maquetación del tema, el *layout* no necesita estructurar nada en estas páginas.

5.8 Atletas y Patrocinadores

Para este caso necesitaremos un *portlet* similar al de las secciones comentadas anteriormente. En un primer acercamiento podría parecer que para cada atleta o patrocinador que se quiera mostrar necesitaremos un *portlet*, pero se trata de una única funcionalidad que lista todos los resultados de una consulta.

En el caso de los atletas podemos tener una cantidad importante de atletas y necesitar una navegación para los resultados actuando de una forma similar al *portlet* de noticia utilizado en las secciones anteriores.

El caso de los patrocinadores es similar pero con menos resultados, seguramente no necesitará de la navegación por los resultados pero no está de más. En este caso hay que tener en cuenta que solo se quieren mostrar patrocinadores activos ya que podemos darlos de baja pero seguimos queriendo tener sus datos almacenados, así pues a la hora de hacer la consulta habrá que añadir una condición para solo recopilar los patrocinadores activos.

Volvemos a contar con solo un *portlet* en cada página.

5.9 Eventos

De las funcionalidades propuestas para el portal por el cliente, esta es seguramente la más específica por lo tanto la más complicada de cubrir. Como ya se ha comentado unifica múltiples tareas en una sola.

Hay dos tipos de eventos: los activos y los no activos. Esta diferenciación parece de poca relevancia pero debe hacerse ya que según esta clasificación los trataremos de una forma u otra.

En la sección de 'Eventos' y dada la clasificación comentada nos encontramos con dos posibles *portlets*. Uno mostrará los eventos activos y otro los inactivos. Ambos tienen en común una funcionalidad pero el *portlet* de eventos activos añadirá otras acciones y posibilidades.

Eventos inactivos

Teniendo en cuenta el diseño de la base de datos que se desarrollo de los requerimientos, el proceso de actividad de un evento se puede considerar desde su creación hasta su fecha. Una vez superada esta fecha el proceso pasara a ser no activo o inactivo.

Del mismo modo que se propone una corrección a lo propuesto con los usuarios cabe decir que si se incluyera un campo de activo como en el caso de las noticias o se especificara un plazo de inscripción para el evento el modelo sería más eficiente. Se pueden usar otros sistemas para controlar esto pero se comentarán en el apartado de eventos activos.

Los eventos, entre otras cosas, deben permitir la visualización de los participantes en el mismo. Seguramente es el mayor atractivo de un evento ya inactivo para el usuario del portal. Por lo tanto el

principal cometido de un *portlet* que se encargue de los eventos inactivos es el presentar la información de este evento.

El *portlet* en cuestión se encargará de consultar la base de datos para obtener la información y listarla. Hay que tener en cuenta que llegados a cierto número de resultados de nuestra consulta deberemos indexarlos y mostrarlos de forma ordenada.

Para esto haremos varias consultas a la base de datos. En la primera seleccionaremos todos los datos de los eventos con fecha inferior a la fecha actual. Una vez tengamos los resultados procederemos a procesarlos de cara a mostrarlos. El resultado vendrá dado en un tipo de datos *ResultSet* así pues cada resultado tendrá que ser *parseado* en el orden de la consulta de cara a obtener cada dato de información para poder mostrarlo.

```
List <Eventos> listaEve = new ArrayList <Eventos> ();
List <String> corredores = new ArrayList <String> ();

ResultSet eventos = BD.obtenerSQL(consulta);

(...)

while(eventos.next()){
    int a = Integer.parseInt(notis.getString(1));

    String consCor = "SELECT corredor.nombre, (...) FROM (...)
    WHERE participa_corr.id_evento='" + a +  "'";
    ResultSet c = BD.obtenerSQL(consCor);
        while(c.next()){
            String aux1 = new String (c.getString(1));
            String aux2 = new String (c.getString(2));
            (...)

            corredores.add(new String(aux1 + ", " + aux2 + (...) ));
        }
    listaEve.add(new Evento(corredores,eventos.getString(2),
        eventos.getString(3),eventos.getString(4), eventos.getString(

    }
```

Figura 5.12 - Fragmentos de código de los eventos inactivos para su visualización.

Para conseguir todos los datos que necesitamos deberemos extraer para cada resultado el *id* del evento y hacer una nueva consulta para obtener los participantes. La forma más interesante de proceder es crear una lista de objetos Evento que instanciaremos con los resultados de nuestras consultas.

De este modo (figura 5.12), anidando dos bucles *While* podremos almacenar en una lista los eventos para poder presentarlos en una JSP. Necesitaremos hacer N+1 consultas a la base de datos (N es el número de eventos resultantes de la primera consulta donde obtenemos todos los eventos inactivos). La primera consulta la ordenamos por fecha de modo que los eventos dentro de la lista ya estarán ordenados de cara a su presentación.

Tener en cuenta también que la consulta de participantes será por partida doble (o triple) ya que debemos comprobar tanto los usuarios no socios como los socios y/o los alumnos, haciendo la consulta algo más complicada.

Una vez hecho esto, se seleccionarán los primeros objetos de la lista eventos y se presentarán en un formato cómodo de lectura. Cada vez que se haga *render* se procederá a acceder a la base de datos, dejando como acción del *portlet* el navegar por los eventos en el caso de que haya demasiados resultados a mostrar (de la forma en la que se presentan las noticias en algunos blogs o los productos en las tiendas *on line*).

Eventos activos

El caso de los eventos activos a primera vista es muy similar al anterior. En un principio no se prevé que vayan a haber más de un evento activo a la vez pero la lógica utilizada para el caso anterior no se modificará. De este modo con un solo cambio en la condición de la consulta para listar solo los eventos con fecha posterior o del día de la consulta tendremos el lado del *render* del *portlet*.

A demás de este cambio (y los de la presentación y título para distinguirlos) necesitamos añadir un par de botones a cada evento que mostremos que distinguirán las dos vistas (o posibles acciones) que tendrán. Estos son los que permitan al usuario apuntarse al evento ya sea como voluntario o como corredor.

En el caso que se decidiera la inclusión de eventos no competitivos en los que no hiciera falta una distinción entre como participar habría que añadir algo de lógica (además de por lo menos otro campo en la base de datos) según los datos de la consulta para desactivar botones. Esto por otro lado podría modificarse con un *Hook*.

Comentar también que como se ha dicho anteriormente hay formas, fuera de modificar las tablas en la base de datos, para poder controlar la inscripción a eventos. Por ejemplo como ya se ha comentado en el caso de los eventos no competitivos (y usando recursos de *Liferay*) sería modificando las *JSP* de este *portlet* mediante *Hooks* haciendo desaparecer los botones, dejando el evento activo pero sin posibilidad de inscripción.

Fase de acción:

Ya hemos visto como actúa el *portlet* en su fase de *render* ya que se comporta de forma muy similar al de eventos inactivos. En cambio su fase de acción es distinta. Compartirá la funcionalidad de listado múltiple con el anterior pero le añadimos nuevas *views* que permiten la inscripción y harán una acción diferente.

Las dos nuevas vistas son dos *JSPs* con un formato de formulario donde habrá que rellenar los datos para la inscripción. Al mandar estos datos haremos la acción y se actualizara la base de datos. Del mismo modo, al terminar esta se pasará a la fase de *render* de todos los *portlets* que contenga la página de eventos en cuestión y si nos inscribimos como corredor nuestros datos ya deberán aparecer en el evento en cuestión.

Participar

Para el caso de los asociados deberemos hacer un par de cambios. En primer lugar si está registrado debemos tener en cuenta a la hora de insertar su participación ya que no es como en el caso anterior, según el tipo de usuario deberemos insertar datos en una u otra tabla.

En este mismo sentido si el corredor es un socio registrado simplemente con una comprobación de si está autenticado en el portal en ese momento pasaremos a bloquear los campos del formulario (esto lo podemos hacer desde el mismo *JSP* o con *Javascript* indistintamente) y permitirle inscribirse (podemos resaltar el hecho de que tendrá que pagar la cuota de carrera).

También podemos optar por el uso de *JavaBeans* de una forma poco habitual para un formulario. Normalmente se utilizan para obtener todos los campos del formulario a la vez, pero en este caso teniendo los datos del registrado podemos rellenar los campos del formulario automáticamente. No habrá una nueva inserción de datos en la tabla de socios ni en la de corredores así que además también habría que bloquear la modificación de los cambios.

En caso contrario, cuando tengamos a un corredor pediremos sus datos y antes de inscribirlo deberemos comprobar que sus datos son correctos y coherentes. Primero veremos si su DNI, que debe ser único, no exista en las tablas de socio o alumno y si existe que sus datos sean todos los mismos (al menos los únicos). En el caso de que no exista procederemos a insertarlo (después de pasar la validación de datos en cuestión).

```
dni = document.getElementById('dni').value;
numero = dni.substr(0,dni.length-1);
let = dni.substr(dni.length-1,1);
numero = numero % 23;
letra='TRWAGMYFPDXBNJZSQVHLCKET';
letra=letra.substring(numero,numero+1);
if (letra!=let.toUpperCase())
{
    document.getElementById('dni').style.backgroundColor='#FF0000';
    cadena=cadena+"Letra del DNI incorrecta.\n";
    if(!focus){
    document.getElementById('dni').focus();
        focus=true;
    }
}
else {
    document.getElementById('dni').style.backgroundColor='#FFFFFF';
}
```

Figura 5.13 - Código javascript para comprobación de veracidad de los datos de DNI introducidos.

Por otro lado este corredor puede haber participado anteriormente en algún evento y puede que sus datos ya estén en la tabla de corredores. En este caso deberán comprobarse los datos que no deberían variar (además del DNI que ya se usa para poder hacer esta comprobación y que ya se ve que esta repetido correctamente) como son nombre, apellidos o fecha de nacimiento.

En el caso de que estos datos no correspondan se deberá presentar un error y si corresponden se debería comprobar si algún otro dato del formulario de los que pueden ser variables (teléfono, dirección, correo electrónico) ha cambiado y actualizarse la base de datos con estos posibles cambios, además de las inserciones en la tabla de participación.

La validez de los datos así como la veracidad del DNI se tratan antes con *Javascript*. Evidentemente no tenemos acceso a la base de datos del gobierno con lo que solo podremos comprobar que los datos sean lógicos.

Colaborar

En el caos de participar como voluntario, tal y como se ha diseñado la base de datos, actuaremos de forma similar al caso anterior pero sin comprobar la existencia de estas personas en las tablas de socio o alumno. Como ya se ha comentado hay varias formas de solucionar el problema de duplicidad que genera pero las especificaciones eran estas.

En todo caso si se desarrolla la funcionalidad de esta forma se puede añadir una pequeña comprobación de si el usuario está registrado, comparando los datos que quiere dar con los reales y mostrando si se desea en caso de disparidad un mensaje e impidiendo la inserción.

Con la solución propuesta en el capítulo de diseño todo la parte de formularios no sería necesaria ya que para poder inscribirse haría falta estar registrado y autenticado y simplemente mostrando los datos, si se desea, antes de proceder a las inserciones en la base de datos (y añadiendo un mensaje sobre el pago de la cuota por participar) tendríamos el *portlet* hecho sin necesidad de validaciones más allá de la comprobación de estar o no ya inscrito.

Podemos presentar otra solución que permitiría llevar a cavo estas acciones y paliaría el problema de las duplicidades. En este caso con añadir unas nuevas tablas intermedias en la que el socio o alumno puede participar como voluntario también y con las comprobaciones de existencia en todo la base de datos antes de insertar nuevos usuarios en ella podríamos solucionarlo. Dejando en el aire el hecho de si un socio o alumno puede ser para el mismo evento voluntario y corredor a la vez. Dependiendo de las funciones de los voluntarios, o algunas de ellas podría ser posible que hiciera ambas cosas aunque no parece ser lo que se quiere.

Visualización

Finalmente comentar la estética. Hasta el momento hemos visto como interactuaría la lógica con los datos tanto desde la base de datos como desde el usuario. También hemos tocado el apartado de validación y formatos (presentación de los datos en *JSPs*).

Finalmente proponemos una posible visualización de la página de Carreras (eventos) en el portal. Por lo que se refiere a temas no hay ninguna necesidad de utilizar uno distinto al del resto del portal y los cambios de colores que se quieren presentar serán los errores en formularios y puede hacerse desde el mismo *javascript* con las validaciones.

La página en si contará con dos *portlets*: Eventos activos y Eventos no activos. En deferencia a la estructuración podemos utilizar un *layout* con dos columnas iguales presentando a la izquierda el *portlet* de Eventos activos y a la derecha el de Eventos inactivos (de forma que según el modelo de lectura occidental encontramos primero los activos y después los inactivos).

Finalmente solo queda la sección de contacto pero se trata de una página con información estática sin mayor relevancia. Puede almacenarse en la base de datos y obtenerse mediante consulta pero se esperan pocos cambios y no es necesario.

6 Conclusiones

6.1 La herramienta y el problema

Opinión y crítica

En el análisis se ha comprobado que *Liferay* trabaja con un gran número de herramientas y estás están hechas usando múltiples lenguajes. Lo que no se ha explorado de forma exhaustiva es el hecho de que acepta el uso de otros tantos *frameworks* y tecnologías. En la introducción se destacan varios casos como pueden ser *Struts*, *Spring, PHP* o *RUBY*.

Esto no forma parte de los objetivos pero es interesante de reseñar y ayuda de cara a ilustrar mejor una de las primeras reflexiones que se presenta: la diversidad. Forma parte del pensamiento de las herramientas *Open Source* y con *Liferay* se explota a grandes dimensiones, si se conoce una tecnología puede aplicarse. Lo que en principio es una ventaja no siempre es así. La evolución de la herramienta ha crecido de esta forma y ha aumentado su complejidad. Para trabajar en desarrollo se necesita conocimientos de muchos campos y para lograr hacer algunas cosas se necesita además cambiar la forma de ver y hacer previa al uso de *Liferay*.

La curva de aprendizaje es muy pronunciada ya que, además de tener que acostumbrarse a la estructuración y aprender a usar algunas de las tecnologías propias de *Liferay* (como los *Hooks*), también se necesitan conocimientos de otras varias si se quiere saber que se hace exactamente. Contando que el hecho de poder trabajar con *frameworks*, beneficia a aquellos que los saben utilizar de antemano pero limita el potencial, la velocidad de desarrollo y el resultado de los que no tiene experiencia con estos. Siempre se puede aprender a utilizar un nueva herramienta para integrarla pero se añade un poco más de complejidad.

La herramienta es bastante generalista y permite tocar varias facetas no solo el desarrollo del portal (por ejemplo se pueden gestionar documentos). Dentro de la comunidad se pueden encontrar personas que han optado por integrar otras herramientas (en el ejemplo de gestión de documentos podría utilizarse *Alfresco*) más especificas e intensivas que funcionan mejor. El hecho de ser *Open Source* hace que esto sea una opción más que valida pero vuelve a añadir más dificultad al uso de *Liferay*. En casos de manejar un gran volumen de datos o tener una gran cantidad de visitas en momentos puntuales, puede ser crítico y requerir de esta práctica.

Resumiendo, la gran cantidad de funcionalidades, herramientas y tecnologías que ofrece *Liferay* permite desarrollar prácticamente lo que sea pero es un arma de doble filo ya que está acompañada de una dificultad y aprendizaje elevados.

Otra de las virtudes actúa de una forma similar, esta es el soporte. En el caso de la versión *Community* (que es la que se ha analizado por motivos principalmente económicos), se puede encontrar mucha documentación de prácticamente todas las versiones y mucha gente trabaja con la herramienta. Esto produce gran cantidad de ideas y posibles ayudas, en algunas ocasiones demasiadas, complicando en momentos concretos encontrar lo que se necesita con exactitud generando una situación algo paradójica en el que tenemos una saturación de información.

Es necesario entrar en una particularización de cara a remachar este apartado. Principalmente el hecho de haber llegado a la conclusión de que es muy recomendable (seguramente ideal) contar con una máquina limpia de contenidos y extensiones *JAVA* y seguramente de otras herramientas de desarrollo porqué aunque no sea muy común, en ocasiones hacen que *Liferay* no funcione correctamente e impida el poder desarrollar.

El segundo problema es el consumo de recursos. Dependiendo de la inversión que se quiera hacer de cara a la creación del portal puede ser realmente un tema complicado. No será problemático si se espera hacer una inversión en cuanto a servidores y a maquinaria para el desarrollador, aunque en general no es el caso del usuario de la versión *Community*.

Por último realzar que nos permite añadir pequeñas funcionalidades o aplicaciones desarrolladas por otros de una forma rápida y sencilla (sobre todo gracias a que los *portlets* están estandarizados). Esto le añade más atractivo y permite que el portal siga evolucionando sin necesidad de dedicarse el propietario a su desarrollo de forma directa, solo necesita aprovechar el trabajo de otros miembros de la comunidad.

Aplicación al mundo de los clubes deportivos

La impresión que causa *Liferay* de cara al desarrollo es que está pensada para creación de portales de organizaciones del mundo laboral. Su sistema de usuarios jerarquizados y agrupables es ideal para este entorno, del mismo modo que su amplio elenco de soluciones de comunicación interna y redes sociales también beneficia este formato de portal.

Esto no tiene porqué ser contraproducente en el caso de un portal de una o varias entidades deportivas. No es lo que se busca concretamente pero no desentona. En estos casos la impresión es que el establecer una jerarquía o una agrupación de usuarios no es tan importante aunque se pueden encontrar casos en los que resultaría útil. Por ejemplo en el caso de que el portal ofrezca la posibilidad de que entrenadores manejen ciertas herramientas para ayudar y planificar el trabajo de los atletas a los que asesora y entrena. Este es un formato parecido en el que hay una.

El trabajar con *portlets* permite poder mostrar en un solo contenedor a la vez varias aplicaciones dinámicas de forma simultánea, en el caso que nos ocupa podría mantener a la vez un marcador actualizado de resultados y un parte meteorológico, ambos ejemplos interesantes en estos casos.

En el ejemplo concreto que hemos conseguido gracias al 'cliente' tenemos un portal bastante sencillo en cuanto a funcionalidades. Esto lleva a pensar que realmente *Liferay* no es la mejor solución para el desarrollo de este portal. Como ya se ha comentado el aprendizaje y la abstracción son muy elevados y se puede llegar a crear un ámbito de trabajo enorme y muy potente.

Cuando se quiere crear un portal que no deje de ser una pequeña comunidad de deportistas y tener además una herramienta de control de la masa social no es preciso el uso de una herramienta de esta envergadura. En el caso de buscar un portal con unas funcionalidades más extravagantes y/o querer hacer que se comporte como un cruce entre estas funcionalidades y otras (plataforma de marketing en la Red o una organización altamente profesional) sería una herramienta mucho más adecuada.

Con *Liferay* se puede hacer prácticamente cualquier tipo de portal pero el esfuerzo que conlleva el aprendizaje no siempre lo convierte en la herramienta más adecuada. Por otro lado si ya se conoce la herramienta y se ha trabajado previamente con esta posiblemente sea una de las opciones más interesantes.

6.2 Objeticos conseguidos

Se han planteado cuatro objetivos principales, de los cuales se han conseguido tres:

1. Análisis de herramienta de creación de portales web.
2. Diseño de un portal con requisitos reales e impuestos por un cliente.

3. Interactuar con la comunidad de herramientas *Open source*.

El primero objetivo se ha cumplido, tras elegir una de herramienta, se ha desglosado su funcionamiento y presentado en el capítulo 5. En el apartado anterior, además, se han nombrado sus pros y sus contras.

El segundo objetivo también se ha conseguido. Además se han propuesto algunas mejoras para intentado corregir algunos puntos problemáticos que surgían de las restricciones dadas por el cliente.

El tercer objetivo se ha cumplido parcialmente. Se esperaba poder aportar algo a la comunidad pero la interactuación a venido más dada por el soporte ofrecido a los problemas surgidos. De todos modos se ha hecho uso activo de la comunidad.

6.3 Objetivos no conseguidos

El último objetivo propuesto era: hacer un prototipo de portal.

Este no se ha completado debido a dos factores fundamentales:

1. El nivel de complejidad de la herramienta y del aprendizaje han sido más elevados de lo esperado.
2. La herramienta a dado problemas en su configuración en la máquina de trabajo y durante un tiempo no ha funcionado de la forma deseada.

Este segundo punto ha afectado también la planificación que se había propuesto. Llegado el punto de conseguir un correcto funcionamiento se ha intentado seguir cumpliendo con los hitos aunque estuviera fuera de tiempo pero tras invertir algo más del tiempo estimado para realizar el proyecto no se ha conseguido cumplir con este objetivo.

6.4 Vías futuras

Quedando un objetivo por cumplir esta es el primer punto en este apartado.

Como propuesta se podría buscar un formato de club más complejo y con unos requisitos de club

menos convencionales de cara a seguir probando las prestaciones de la herramienta.

Otra opción es la de analizar *Liferay* dentro de otro campo especifico, como ejemplo ya se ha comentado la opción de desarrollar un portal para una empresa o una organización parece más adecuado.

Por último y siguiendo con el espíritu analítico de este trabajo podría ser de interés ver cómo respondería otra herramienta *Open Source* y hacer en este caso el análisis de esta y llevar acabo también un análisis comparativo entre ambas. Este punto se puede ampliar con tantas herramientas como se deseen analizar.

7 Anexos

7.1 Hoja de requerimientos Portal Atletismo

A. Funcionalidades

Actualmente existe ya un portal y se desea mantener todas las funcionalidades actuales y añadir nuevas.

El portal cuenta con siete secciones:

‘Inici’, ‘Atletes’, ‘Patrocinadors’, ‘Contacte’, ‘Curses’, ‘Escola’, ‘Blog/Historic’.

‘Inici’, ‘Curses’ y ‘Escola’ son básicamente secciones de noticias.

‘Atletes’ y ‘Patrocinadors’ nos dan información sobre estos dos activos del club de atletismo.

‘Contacte’ da la información de contacto y algo de historia sobre el Club.

Finalmente la sección de ‘Blog/Historic’ es una recopilación de todas las noticias en formato blog.

No tenemos información sobre la parte privada del portal actualmente.

Se espera añadir las siguientes funcionalidades:

Parte pública del portal:

La presentación del portal se dividirá en cuatro grandes bloques donde podremos ver las últimas noticias de cada una de las secciones y des de los cuales podremos acceder a estas secciones. Las secciones que se han propuesto son: ‘Pista’, ‘Carretera’, ‘Montaña’ y ‘Escuela’.

Al acceder a una de las secciones veremos tan solo las noticias y datos de esta sección.

Se añadirán nuevos datos con respecto a tiempos (marcas personales) en la sección de atletas que ahora solo cuenta con una fotografía y el nombre y apellidos.

Las noticias, sean del tema que sean, contarán con comentarios para los usuarios. Los comentarios

podrán ser anónimos (ya que puede ser de interés para gente externa al club) pero se buscará implementar validación para impedir mensajes de tipo 'SPAM' automáticos.

Actualmente la página solo aparece en un idioma, catalán. Se busca añadir más idiomas (principalmente castellano e inglés).

Se quiere dotar al portal con un sistema de usuarios para los socios del club. De forma que al inscribirse al club también se pueda crear un perfil de usuario para navegar identificado, como socio, en la web. El socio será capaz de modificar parte de sus datos del perfil, comentar en noticias de forma no anónima e inscribirse en diferentes actividades (principalmente carreras).

En la sección de carreras a parte de noticias tendremos la posibilidad de inscribirnos como voluntarios y/o corredores en los eventos que lo requieran/permitan.

Parte privada:

El/los usuario/s encargado/s de administrar (administrador a partir de ahora) podrán acceder de forma remota, identificándose como administrador, al portal.

Desde la parte privada se podrá acceder a ciertos criterios de configuración, crear y modificar noticias y carreras, gestionar altas y bajas de patrocinadores, tramitar bajas y verificaciones de altas de socios y generar 'reports' sobre socios, carreras y demás datos. Estos informes serán exportables.

B. Datos

Tipos de usuarios del portal:

Socio:

- Nombre y apellidos.
- NIF/NIE.
- Correo electrónico.
- Fecha de nacimiento.
- Dirección del domicilio.
- Teléfono de contacto.
- Número de chip.
- Talla de ropa.
- Número de cuenta corriente.
- Foto.
- Mejores marcas personales.
- Especialidad.
- Nombre de identificación en el portal.
- Contraseña de acceso al portal.

Alumno:

- Nombre y apellidos.
- Fecha de nacimiento.
- Dirección del domicilio.
- Teléfono de contacto.
- Tutor.
- Talla de ropa.
- Número de cuenta corriente.
- Nombre de identificación en el portal.
- Contraseña de acceso al portal.
- NIF/NIE.*
- Correo electrónico.*
- Foto.*

*: El perfil de este usuario es el de un menor y puede que no cuente con algunos de estos datos o no se quiera por parte del club o del tutor revelarlos.

Administrador:

- Nombre de identificación en el portal.
- Contraseña de acceso al portal.

Usuario anónimo.

Otras entidades con datos:

Carrera:

- Nombre del evento.
- Fecha.
- Descripción.
- Categoría.
- Ubicación.
- Cuota.

Voluntario:

- Nombre y apellidos.
- NIF/NIE.
- Correo electrónico.
- Fecha de nacimiento.
- Dirección del domicilio.
- Teléfono de contacto.
- Evento.

Corredor:

- Nombre y apellidos.
- NIF/NIE.
- Correo electrónico.
- Fecha de nacimiento.
- Dirección del domicilio.
- Teléfono de contacto.
- Evento.
- Número de cuenta corriente.

Noticia:

- Titular.
- Categoría.
- Cuerpo de la noticia.
- Imagen.
- Pie de imagen.
- Fecha de publicación.
- Fecha de caducidad.*
- Activa.*

*: Campos opcionales.

7.2 Glosario

API [41]

Interfaz de programación de aplicaciones (en inglés *Application Programming Interface*) es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro *software* como una capa de abstracción. Son usadas generalmente en las librerías representando la capacidad de comunicación entre componentes.

CSS [35]

Las Hojas de Estilo en Cascada (*Cascading Style Sheets*) son un lenguaje para describir la presentación de los documentos. Aparecieron con el estándar de *HTML4* como complemento para definir el aspecto y la presentación que debían tener los documentos estructurados de tipo *HTML*.

CSS permite adaptar un mismo contenido a diversos medios de visualización como son: una pantalla de ordenador, una versión impresa o un intérprete de braille.

Actualmente *CSS* va por la tercera revisión, aunque todavía no es soportada por todos los navegadores. La versión *CSS2* ya está ampliamente extendida y utilizada por cualquier diseñador o maquetador web.

HTML [36]

El formato *HTML* (*HyperText Markup Language*) es el estándar para la navegación en la web. Actualmente se está desarrollando la versión *HTML5* que pretende incorporar video de forma fácil, pero todavía no está soportado por los navegadores existentes.

Liferay como cualquier herramienta web, acaba generando un código en *HTML* que es enviado al navegador del cliente para ser interpretado. Este lenguaje se apoya en los demás para modificar su comportamiento y aspecto visual, porque por sí solo entregaría contenido estático.

IDE [37]

Un *IDE* (sigla de *integrated development environment*) es un entorno de programación que ha sido empaquetado como un programa de aplicación. Consiste en un editor de código, un compilador, un

depurador y un constructor de interfaz gráfica (*GUI*). Los *IDEs* pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

Los *IDE* proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación. En algunos lenguajes, un *IDE* puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto.

(Enterprise) **JavaBeans [38]**

El objetivo de los *EJB* es dotar al programador de un modelo que le permita abstraerse de los problemas generales de una aplicación para centrarse en el desarrollo de la lógica de esta. El hecho de estar basado en componentes permite que éstos sean flexibles y sobre todo reutilizables.

No hay que confundir los Enterprise *JavaBeans* con los JavaBeans. Los JavaBeans también son un modelo de componentes creado por *Oracle - Sun Microsystems* para la construcción de aplicaciones, pero no pueden utilizarse en entornos de objetos distribuidos al no soportar la invocación remota.

JavaScript-J*query* [39]

JavaScript es un lenguaje de programación interpretado por el navegador web. Esta tecnología se ejecuta en el lado del cliente (dentro de su navegador) y es utilizada para reaccionar ante eventos del usuario o para dar dinamismo a las páginas web, como por ejemplo, efectos de deslizarse o de sustitución de imágenes.

Jquery es uno de los *framework* de *Javascript* más extendidos actualmente. Principalmente, por facilitar el acceso a los elementos *HTML* del documento y poder interactuar con facilidad con el aspecto visual que establece el *CSS*, creando efectos visuales muy vistosos que sin *Jquery* conllevarían un desarrollo elevado.

JSP [26]

JSP es el acrónimo de *Java Server Pages* y consiste en una tecnología *JAVA* que permite generar contenido *HTML* o *XML* de forma dinámica, haciendo uso de la máquina virtual de *JAVA* y, pudiendo utilizar las clases y todo el potencial que este lenguaje ofrece. Al tratarse de *JAVA*, *Sun*

Microsystems fue el responsable de los dos estándares que se han generado *JSP 1.2* y *JSP 2.1*, y están estrechamente ligados a los estándares de *servlet* que han aparecido.

JVM[42]

Una máquina virtual *JAVA* (siglas en inglés JVM) es una máquina virtual ejecutable en una plataforma específica, capaz de interpretar código máquina de bajo nivel haciendo de puente entre el hardware y el código *JAVA* que ejecutaremos por encima de la JVM.

SDK [40]

SDK (siglas de *software development kit*) es generalmente un conjunto de herramientas de desarrollo de software que permiten crear aplicaciones para un sistema concreto, por ejemplo ciertos paquetes de software, *frameworks*, plataformas de hardware, sistemas operativos, etc.

Es algo tan sencillo como una interfaz de programación de aplicaciones o API (*Application Programing Interface*) creada para permitir el uso de cierto lenguaje de programación, o puede, también, incluir hardware sofisticado para comunicarse con un determinado sistema embebido.

Velocity [28]

Velocity es un motor generador de estructuras de página, basado en Java. Proporciona un modelo sencillo de referenciar objetos definidos en el código Java. Su propósito es realizar de forma clara y simple la capa de presentación dentro del patrón de diseño "*Model View Controler*", donde se separan claramente las capas de presentación, estructura y control, siendo *Velocity* la capa de presentación para un modelo *MVC* desarrollado en Java.

La gran ventaja que aporta el uso de *Velocity* es la posibilidad de trabajar en paralelo desarrolladores de software y diseñadores web, siempre y cuando ambos mantengan el modelo *MVC*.

XML [27]

XML es un lenguaje de marcado que predomina en la parte de configuración del servidor. Gran parte de los parámetros y valores iniciales de las variables son guardadas en este tipo de documento.

7.3 Comunicación con la Base de Datos.

Para la conexión, consulta e inserción en la base de datos desde Java se ha utilizado una clase en forma de *utility* propia.

```
12  public class BDclub {
13
14      private Connection conexion = null;
15      private Statement sentenciaSQL = null;
16      private ResultSet cdr = null;
17      private MysqlDataSource dataSource = null;
18
19      public BDclub() throws SQLException{
20          dataSource= new MysqlDataSource();
21          conectar();
22
23      }
24
25      public void conectar() throws SQLException {
26          dataSource.setUser("root");
27          dataSource.setPassword("admin");
28          dataSource.setDatabaseName("clubdb");
29          dataSource.setServerName("localhost");
30
31          conexion = dataSource.getConnection();
32          sentenciaSQL = conexion.createStatement();
33
34      }
35
36      public void cerrarConexion() throws SQLException{
37          if(cdr!= null)cdr.close();
38          if(sentenciaSQL != null)sentenciaSQL.close();
39          if(conexion != null)conexion.close();
40
41      }
42
43
44      public ResultSet obtenerSQL(String sent)throws SQLException{
45          return sentenciaSQL.executeQuery(sent);
46      }
47
48      public int updateSQL(String sent)throws SQLException{
49          return sentenciaSQL.executeUpdate(sent);
50      }
51
52  }
```

Figura 7.1 - Clase para la comunicación con la base de datos.

8 Referencias

[1] Club de Petanca Los Infantes - http://cplosinfantes.jimdo.com/

[2] FC Bayern München - http://www.fcbayern.telekom.de/de/splash.php

[3] Panathinaikos BC - http://www.paobc.gr/page.ashx?pid=1

[4] NBA - http://www.nba.com/

[5] NHL - http://www.nhl.com/

[6] NFL - http://www.nfl.com/

[7] LFP - http://www.lfp.es/

[8] Real Federación Española de Patinaje, *OK Liga* http://www.fep.es/website/comp_clasificacion.asp?modalidad=14&par=ok

[9] Fourier Series Calculator – Series de Fourier On Line http://www.mathstools.com/section/main/fourier_series_calculator#.T9m4IFKfba8

[10] DinoRPG - http://es.dinorpg.com/

[11] Wikipedia, *Servlet* - http://es.wikipedia.org/wiki/Java_Servlet

[12] Wikipedia, *Portlet* - http://es.wikipedia.org/wiki/Portlet

[13] OpenPortal - http://openportal.sourceforge.net/

[14] uPortal - http://www.jasig.org/uportal

[15] LIFERAY - http://www.liferay.com/

[16] Youtube, *uPortal Eclipse tutorial* - http://www.youtube.com/watch?v=iOdBd81P1S4

[17] Drupal - http://drupal.org/

[18] OpenText - http://www.opentext.com/2/global.htm

[19] 'Perficient, Inc', *Gartner Magic Quadrant for Horizontal Portals 2011* http://blogs.perficient.com/portals/2011/10/31/gartner-magic-quadrant-for-horizontal-portals-2011/

[20] LIFERAY, *Liferay Portal Downloads* http://www.liferay.com/es/downloads/liferay-portal/available-releases

[21] LIFERAY, *Liferay 6.1 CE ya está disponible* http://www.liferay.com/web/francisco.fernandez/blog/-/blogs/12030057

[22] Init Developers, *Desarrollos con Liferay: más portlets y menos bugs, por favor* http://blog.theinit.com/2011/06/13/desarrollos-liferay-portlets-y-menos-bugs/

[23] LIFERAY, *Quick installation instructions* http://www.liferay.com/community/wiki/-/wiki/Main/Quick+Installation+Instructions

[24] LIFERAY, *IDE installation guide* http://www.liferay.com/community/wiki/-/wiki/Main/Liferay+IDE+Installation+Guide

[25] LIFERAY, *IDE getting started tutorial*
http://www.liferay.com/community/wiki/-/wiki/Main/Liferay+IDE+Getting+Started+Tutorial
[26] Wikipedia, *JSP* - http://en.wikipedia.org/wiki/JavaServer_Pages
[27] Wikipedia, *XML* - http://en.wikipedia.org/wiki/XML
[28] Apache, *Velocity* - http://en.wikipedia.org/wiki/Apache_Velocity
[29] LIFERAY, *Portal Hooks plugins*
http://www.liferay.com/community/wiki/-/wiki/Main/Portal+Hook+Plugins
[30] LIFERAY, *Creating you first EXT plugin*
http://www.liferay.com/web/edward.shin/blog/-/blogs/4701348
[31] LIFERAY, *Layout Template*
http://www.liferay.com/community/wiki/-/wiki/Main/Layout+Template
[32] LIFERAY, *Themes* - http://www.liferay.com/community/wiki/-/wiki/Main/Themes
[33] Liferay Portal 6.0 – Development Guide
http://www.liferay.com/documentation/liferay-portal/6.0/development/-/ai/creating-a-portlet
[34] LIFERAY, *Development guide*
http://docs.liferay.com/portal/6.0/official/liferay-developer-guide-6.0.pdf
[35] Wikipedia, *CSS* - http://es.wikipedia.org/wiki/Hojas_de_estilo_en_cascada
[36] Wikipedia, *HTML* - http://es.wikipedia.org/wiki/HTML
[37] Wikipedia, *IDE* - http://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado
[38] Wikipedia, *Enterprise JavaBeans* - http://es.wikipedia.org/wiki/Enterprise_JavaBeans
[39] Wikipedia, *JavaScript* - http://es.wikipedia.org/wiki/JavaScript
[40] Wikipedia, *SDK* - http://es.wikipedia.org/wiki/SDK
[41] Wikipedia, *API*
http://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones
[42] Wikipedia, *JVM* - *http://es.wikipedia.org/wiki/JVM*

Resumen

El proyecto puede dividirse en dos grandes partes, diseño de un portal deportivo y análisis de una herramienta *Open Source* para el desarrollo de portales web. El apartado de diseño cuenta con un club de atletismo que proporciona unos requerimientos y necesidades de cara a tener un problema real y no basado en especulaciones. Se ha diseñado tanto la base de datos como la estructura del portal y se tienen en cuenta las necesidades del cliente. La parte de análisis de una herramienta *Open Source* desglosa los módulos de esta, viendo que necesidades cubre cada uno y que pueden hacer, que tecnologías usan y que soluciones pueden dar al problema planteado.

Resum

El projecte es pot dividir en dos grans parts, disseny d'un portal esportiu i anàlisis d'una eina *Open Source* per a desenvolupament de portals web. L'apartat de disseny compta amb un club d'atletisme que proporciona uns requeriments i necessitats de cara a enfrontar-se a un problema real i no basar-lo en especulacions. S'ha dissenya tant la base de dades como l'estructura del portal i es ten en compte les necessitats del client. La part d'anàlisi d'una eina *Open Source* desglossa els mòduls d'aquesta, mostrant quines de les necessitats cobreix cadascun i que pot fer, quines tecnologies fa servir i quines solucions aporta al problema plantejat.

Abstract

The Project can be divided in two parts, the design of a sports club portal and the analysis of an Open Source tool for web portals development. The design part gets help from an athletic club that provides the project with requirements and needs in order to cope with a real problem. It is designed both the database as the portal structure and takes into account customer needs. The analysis part of an Open Source tool takes in account the modules of the tool, seeing the needs that cover and the tasks each one can do, the technologies that are used and the solutions provided to the problem.