Universitat Autònoma
de Barcelona

Gestor web de una entidad deportiva

Francisco Javier Aguado Joaquín

18 de septiembre de 2013

El sotasignat, **Raúl Aragonés**, professor de l'Escola d'Enginyeria de la UAB,

CERTIFICA:

Que el treball al que correspon la present memòria ha estat realitzat sota la seva direcció per:

Francisco Javier Aguado Joaquín

I per a que consti firma la present.

Sabadell, Setembre de 2013

En memoria de David Albiñana Macías.

Presidente y fundador del Club de Fútbol Sala Sant Boi.

Resumen

Castellano

El proyecto consiste en el diseño y desarrollo de una aplicación web para la gestión de una entidad deportiva. La aplicación ha de permitir la gestión de las distintas facetas deportivas: temporadas, categorías, ligas, equipos, resultados y clasificaciones. También será un escaparate de lo que ocurre en la entidad para socios y aficionados; así, también ha de servir para la publicación de noticias relacionadas con la entidad y facilitar la información en una zona pública que podrá ver cualquiera.

Català

El projecte consisteix en el diseny i desenvolupament d'una aplicació web per a la gestió d'una entitat esportiva. L'aplicació ha de permetre la gestió de les diferents facetes esportives: temporades, categories, lligues, equips, resultats i clasificacions. També serà un aparador del que succeix a la entitat per a socis i aficionats; així, també ha de servir per a la publicació de notícies relacionades amb l'entitat i facilitar la informació en una zona pública que podrà veure qualsevol.

English

The project consists to designing and developing of a web application for managing a sports organization. The application must allow the management of these sports facets: seasons, categories, leagues, teams, results and standings. Also will showcase what happens in the organization for members and fans, and also must be useful for the publication of news related to the organization and provide the information in a front end that anyone can see.

Índice general

1. Introducción **7**
- 1.1. Presentación y objetivos . . . 8
- 1.2. Contexto actual . . . 8
- 1.3. Motivaciones personales . . . 8
- 1.4. Estructura de la memoria . . . 9

2. Estudio de viabilidad **10**
- 2.1. Introducción . . . 11
  - 2.1.1. Tipología y palabras clave . . . 11
  - 2.1.2. Descripción . . . 11
  - 2.1.3. Objetivos del proyecto . . . 11
  - 2.1.4. Partes interesadas . . . 12
- 2.2. Estudio de la situación actual . . . 12
  - 2.2.1. Contexto . . . 12
  - 2.2.2. Lógica del sistema . . . 13
  - 2.2.3. Usuarios y/o personal del sistema . . . 13
  - 2.2.4. Diagnóstico del sistema . . . 13
  - 2.2.5. Normativas y legislación . . . 14
- 2.3. Requisitos del sistema . . . 14
  - 2.3.1. Requisitos funcionales . . . 14
  - 2.3.2. Requisitos no funcionales . . . 14
  - 2.3.3. Restricción del sistema . . . 14
  - 2.3.4. Catalogación y priorización de los requisitos . . . 15
- 2.4. Alternativas . . . 15
  - 2.4.1. Sage Eurowin Entidades deportivas . . . 15
  - 2.4.2. Fly Solutions . . . 16
- 2.5. Selección de la solución . . . 16
- 2.6. Conclusión . . . 16

3. Planificación **17**
- 3.1. Fases y actividades del proyecto . . . 18
- 3.2. Recursos del proyecto . . . 18
- 3.3. Calendario del proyecto . . . 19
  - 3.3.1. Calendario de los recursos . . . 19
  - 3.3.2. Cuadro de tareas del proyecto . . . 20
  - 3.3.3. Diagrama de Gantt . . . 21
- 3.4. Coste del proyecto . . . 21
- 3.5. Evaluación de riesgos . . . 22

3.5.1. Lista de riesgos . . . 22
3.5.2. Catalogación de los riesgos . . . 23
3.5.3. Plan de contingencia . . . 23

4. Fundamentos teóricos **24**
4.1. Aplicaciones web . . . 25
4.1.1. ¿Qué es una aplicación web? . . . 25
4.1.2. Estructura de una aplicación web . . . 25
4.1.3. Lenguaje HTML . . . 26
4.1.4. Hojas de estilo en cascada o CSS . . . 27
4.1.5. less CSS . . . 27
4.1.6. JavaScript . . . 30
4.1.7. jQuery . . . 31
4.1.8. Boostrap . . . 32
4.1.9. PHP . . . 32
4.2. Bases de datos . . . 34
4.2.1. MySQL . . . 34
4.2.2. Entidad - Relación . . . 35
4.2.3. Configuración de una entidad . . . 35
4.3. MVC o Modelo-Vista-Controlador . . . 36
4.3.1. Controlador (Controller) . . . 37
4.3.2. Modelo (Model) . . . 37
4.3.3. Vista (View) . . . 37

5. Herramientas utilizadas **39**
5.1. LAMP o Linux-Apache-MySQL-PHP . . . 40
5.1.1. Apache . . . 40
5.1.2. MySQL . . . 41
5.1.3. PHP . . . 41
5.2. phpMyAdmin . . . 41
5.3. NetBeans . . . 42
5.4. Git . . . 43
5.4.1. GitHub . . . 44

6. Análisis y diseño **46**
6.1. Base de Datos . . . 47
6.1.1. Diseño entidades . . . 47
6.1.2. Diagrama relacional . . . 55
6.2. Zona de administración (backend) . . . 55
6.2.1. Usuario no autentificado . . . 55
6.2.2. Usuario con rol editor . . . 57
6.2.3. Usuario con rol entrenador . . . 61
6.2.4. Usuario con rol resultados . . . 62
6.2.5. Usuario con rol planificador . . . 66
6.2.6. Usuario con rol administrador . . . 77
6.3. Estructura del código en el Back-End . . . 81
6.3.1. application . . . 81
6.3.2. cfs . . . 82
6.3.3. database . . . 83
6.4. Desarrollo de los módulos . . . 83

6.4.1. Estructura básica de un módulo 83
6.4.2. Módulo home 84
6.4.3. Módulo temporadas 84
6.4.4. Módulo categorias 85
6.4.5. Módulo ligas 85
6.4.6. Módulo equipos 86
6.4.7. Módulo jornadas 88
6.4.8. Módulo resultados 88
6.4.9. Módulo clasificaciones 89
6.4.10. Módulo imagenes 89
6.4.11. Módulo noticias 90
6.4.12. Módulo usuarios 91
6.4.13. Módulo entrenadores 91
6.4.14. Módulo notificaciones 92
6.5. Zona pública 93

7. Pruebas **96**
7.1. Introducción 97
7.1.1. Pruebas unitarias 97
7.1.2. Pruebas de integración 97
7.1.3. Pruebas de compatibilidad 97
7.1.4. Errores detectados 101

8. Conclusiones **103**

9. Líneas futuras **105**

10. Bibliografía y Referencias **107**
10.1. Bibliografía 108
10.2. Referencias 108

1. Introducción

1.1. Presentación y objetivos

El proyecto consiste en un Sistema de Gestión de Contenidos (Content Management System o por sus siglas, CMS, en inglés) orientado a un club de fútbol sala. El club en el que se centra el desarrollo es el Club de Fútbol Sala Sant Boi, del cual formo parte como junta directiva. Si bien es cierto que la aplicación es fácilmente adaptable a cualquier otro club, tanto de fútbol sala como fútbol campo.

La idea nace por la necesidad de no depender de alguien con conocimientos de programación web cada vez que se quiera actualizar la información ofrecida en la página. Entre dicha información se encuentran los equipos, partidos, resultados y clasificaciones. Así, cada vez que se actualicen los resultados en la parte interna de la web, estos estarán visibles para todos los socios y seguidores del equipo en la parte pública. También estarán accesibles las fotos de los equipos y noticias que puedan interesar.

1.2. Contexto actual

El club lleva ya años en activo. Fue fundado en 1999 así que ha ido generando una pequeña gran masa social que desea estar informada sobre lo que ocurre relacionado con la entidad. Por otro lado, cada año vamos consiguiendo tener más equipos; equipos que tienen, de media, 10 jugadores. Estos jugadores, junto con amigos y familiares, son los primeros en querer estar informados de qué ocurre; para acto seguido compartir la información con el resto de familiares y amistades en las redes sociales[1]. Esta acción, hoy en día, está en la mano de todo el mundo gracias a los smartphones[2].

Debido a lo expuesto, se me fue asignada la tarea de creación de un sitio web donde poder consultar la información. Creo que como entidad tenemos la necesidad y el deber de informar con facilidad. La necesidad porque ha sido gracias a que la gente nos conoce que se ha podido crecer y seguir creciendo. El deber dado que una vez dejas formar parte de la entidad, aunque sea como mero aficionado, tienes el deber para esa persona.

1.3. Motivaciones personales

Siempre me han apasionado las tecnologías web. Así, en el momento que se me encomendó dicha tarea, no pude negarme. La verdad es que hacer una aplicación de estas características no es nada sencillo, pero me apetecía. Quería poner a prueba todo lo aprendido y ver si era capaz de realizar una solución a un problema puntual.

Por otro lado, existe una motivación emocional. El antiguo presidente, que por desgracia ya no está entre nosotros, delegó el funcionamiento del club en nosotros; unos chicos recién salidos del instituto. Creo que lo estamos haciendo bien pero, por David (así se llamaba), me gustaría que fuéramos un referente

[1] Aplicativos, ya sea web o móvil, en el que compartir experiencias, fotografías y vídeos con los amigos

[2] Teléfono inteligentes que no solo realizan la función típicamente asignada a un dispositivo; sino que, prácticamente, son un ordenador en al mano

y para ello, hoy por hoy, hace falta que la gente te conozco; es decir, estar en la red.

1.4. Estructura de la memoria

En la **introducción** se incluye la presentación donde se explica la idea principal, una explicación del contexto y las motivaciones para realizar el proyecto.

En el apartado de **estudio de viabilidad** tenemos una introducción al estudio, los objetivos deseados, explicación de la situación actual, las especificaciones, la planificación seguida, la valoración de alternativas y riesgos. También se explican los diferentes riesgos que puedan aparecer. Finalmente, las conclusiones del estudio.

Dentro de **fundamentos teóricos** se explicarán todos los aspectos necesarios para la comprensión del proyecto y su implementación.

En el **análisis** se explican los diferentes puntos a tener en cuenta a la hora de implementar la aplicación. Diferentes escenarios en los que se puede encontrar la aplicación.

Dentro del apartado **implementación** se detalla la ejecución del proyecto.

El apartado **pruebas** es un listado de las diferentes pruebas que se han realizado para asegurar el buen funcionamiento del software.

Las **conclusiones** son una reflexión sobre cómo se ha desarrollado el proyecto.

Para terminar, en el apartado **bibliografía** tenemos toda la documentación consultada.

2. Estudio de viabilidad

Todo gran proyecto necesita un estudio exhaustivo sobre si será viable para las necesidades requeridas y sobre los recursos necesarios. Se trata del primer paso para iniciar el desarrollo con la seguridad de poder llegar a buen puerto.

2.1. Introducción

2.1.1. Tipología y palabras clave

La tipología es desarrollo. Este proyecto puede estar definido por las siguientes palabras claves: gestor, administración, deporte, PHP[3], MySQL[4], JavaScript[5], JQuery[6], HTML[7], CSS[8], less[9], Bootstrap[10], Framework[11], Git[12], Firebug[13].

2.1.2. Descripción

Existen muchos clubes deportivos en este país. Hay un gran vacío a la hora de poder gestionar todo el tema referente al mismo: socios, jugadores, entrenadores, cuentas, subvenciones, noticias, clasificaciones, partidos... Es muy importante, y más hoy en día, tener la posibilidad de poder gestionar un club por diversas personas desde distintos puntos geográficos al mismo tiempo.

2.1.3. Objetivos del proyecto

1. **Gestión de usuarios**: administrador, editor, planificador...
2. **Gestión de competiciones**: equipos, enfrentamientos, resultados...
3. **Publicación de noticias**

Objetivo	Crítico	Prioritario	Secundario
1	X		
2	X		
3	X		

Tabla 2.1 Objetivos del proyecto

[3]lenguaje ejecutado de la parte del servidor, orientado a aplicaciones web.
[4]servidor de base de datos.
[5]lenguaje de programación interpretado. Principalmente se utiliza en su forma del lado del cliente permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.
[6]Biblioteca de JavaScript.
[7]HyperText Markup Language (lenguaje de marcado de hipertexto). Lenguaje utilizado para describir y traducir la estructura e información de una página web.
[8]Cascading Style Sheets u hojas de estilo en cascada, sirven para dar forma y color a la programación en HTML.
[9]Se trata de un lenguaje que extiende a CSS con la posibilidad de añadir variables y funciones
[10]Framework CSS para la realización rápida de sitios web.
[11]es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software.
[12]Sistema de control de versiones; básicamente utilizado para controlar proyectos de programación.
[13]Extensión para el navegador web Mozilla Firefox que nos permite inspeccionar HTML, CSS y JavaScript en tiempo de ejecución.

2.1.4. Partes interesadas

En la tabla (Tabla 2.2) podemos ver las partes interesadas que participan en el proyecto.

Partes interesadas (stakeholders)		
Fco. J. Aguado.	Desarrollador/Analista.	Desarrollará el proyecto.
Raúl Aragonés.	Jefe de Proyecto.	Supervisará el proyecto.
Junta directiva.	Usuarios web.	Beta[14] testers.

Tabla 2.2 Partes interesadas

En la siguiente tabla (Tabla 2.3) podemos ver un listado de los roles que desempeñarán los usuarios en nuestra aplicación.

Partes interesadas (perfiles de usuario)		
Admin.	Administrador del sistema.	Acceso total.
Editor.	Sección noticias.	Crear, editar y eliminar noticias.
Planificador.	Administrador acotado.	Acceso total para planificar la temporada: categorías, ligas, equipos...
Resultados.	Miembro club.	Accederá para introducir resultados de partidos.
Entrenador.	Entrenador de un equipo.	Solo tendrá acceso a su equipo.

Tabla 2.3 Roles en la aplicación

En la última tabla de este apartado (Tabla 2.4) tenemos las personas que componen el equipo de proyecto.

Partes interesadas (Project team)		
Fco. J. Aguado.	Desarrollador/Analista.	Desarrollará el proyecto.
Raúl Aragonés.	Jefe de Proyecto.	Supervisará el proyecto.
Junta directiva.	Miembro club.	Beta[14] testers.

Tabla 2.4 Equipo del proyecto

2.2. Estudio de la situación actual

2.2.1. Contexto

Hay una aplicación web para el tema de resultados y noticias, pero no hay una gestión adecuada de usuarios y faltan muchas características deseadas. A

[14]fase de la aplicación donde se hacen las pruebas finales antes de lanzarla a producción

parte, el diseño interno de la aplicación no es nada mantenible y muy complicada agregar funcionalidades nuevas.

2.2.2. Lógica del sistema

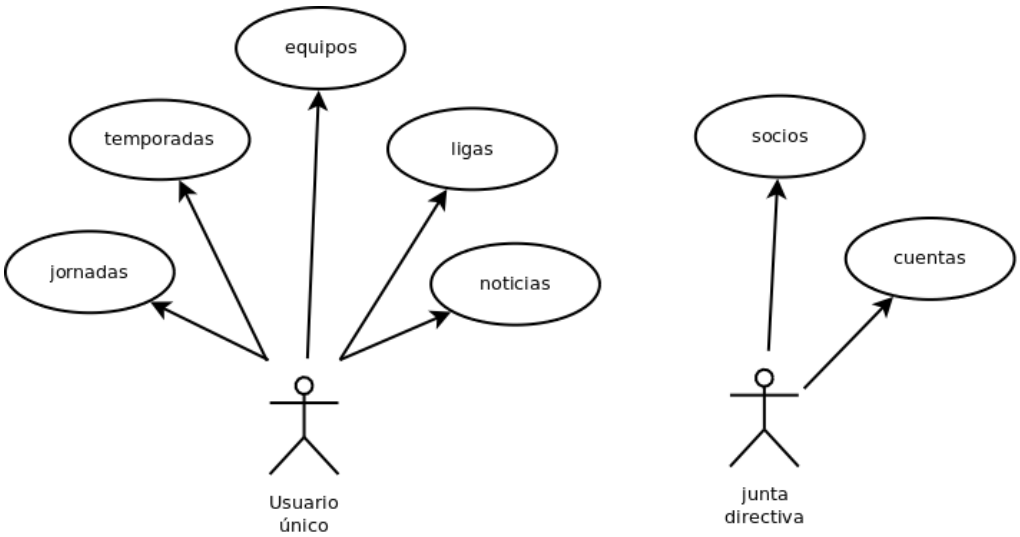

Figura 2.1 Lógica actual del sistema.

2.2.3. Usuarios y/o personal del sistema

En la tabla (Tabla 2.5) podemos ver la organización del sistema actual.

Partes interesadas (Project team)	
Administrador.	Lo hace todo en la web. Ahora mismo es el vicepresidente.
Contable.	Hace las cuentas en un Excel[15].

Tabla 2.5 Partes interesadas del sistema actual

2.2.4. Diagnóstico del sistema

Deficiencias:

- El sistema actual es inseguro.
- Un solo usuario lo realiza todo, hay un usuario y es administrador; no hay más.

Mejoras:

- Acceso deslocalizado a la gestión de la entidad.

[15] Aplicación que permite manipular datos numéricos y alfanuméricos dispuestos en forma de tablas compuestas por celdas.

- Poder repartir el trabajo entre varios para no saturar a una persona sola.
- Poder guardar y recuperar (en un futuro) estadísticas de temporadas anteriores.

2.2.5. Normativas y legislación

- LOPD[16].

2.3. Requisitos del sistema

2.3.1. Requisitos funcionales

1. Administrar los permisos de los usuarios dependiendo del rol[17] asignado a dicho usuario.
2. Los usuarios con rol administrador podrán modificar cualquier dato.
3. Los entrenadores, por el simple hecho de serlo, serán administrador de su equipo.
4. Los usuarios con permiso de editor podrán publicar noticias que se verán en la portada pública.
5. Los entrenadores podrán planificar la temporada de su equipo poniendo los enfrentamientos.
6. La entidad podrá organizar torneos de verano.
7. Los usuarios con rol Planificador podrán añadir, editar y borrar. temporadas, categorías, ligas, equipos y jornadas.

2.3.2. Requisitos no funcionales

1. Cumplimiento de la LOPD[16] en cuanto a almacenar en territorio nacional los datos.
2. Cada acción realizada será registrada por si hubieran problemas.
3. La aplicación ha de ser fácil de aprender.

2.3.3. Restricción del sistema

Ahora mismo el sistema no tiene ninguna restricción.

[16]Ley Orgánica de Protección de Datos
[17]Un rol es una función que se ejerce y que conlleva ciertos privilegios y restricciones
[16]Ley Orgánica de Protección de Datos

2.3.4. Catalogación y priorización de los requisitos

	RF1	RF2	RF3	RF4	RF5	RF6	RF7
Esenciales	X	X		X			X
Condicional			X				
Opcional					X	X	

Tabla 2.6 Requisitos funcionales

	RNF1	RNF2	RNF3
Esenciales	X	X	
Condicional			X
Opcional			

Tabla 2.7 Requisitos no funcionales

	RF1	RF2	RF3	RF4	RF5	RF6	RF7	RNF1	RNF2	RNF3
O1	X	X	X	X			X	X	X	X
O2		X	X		X	X	X	X	X	X
O3		X		X				X	X	X

Tabla 2.8 Requisitos y objetivos

2.4. Alternativas

Existen varias alternativas pero debido a los presupuesto y características que manejan los pequeños clubes, queda fuera de su alcance u ofrecen demasiadas funcionalidades que luego no se van a utilizar.

2.4.1. Sage Eurowin Entidades deportivas

Sage es una empresa dedicada a desarrollar utilidades de gestión para diferentes sectores. Realmente es un software de gestión integral que tiene opciones para varios sectores, no es específico.
Funcionalidades:

- Contabilidad.
- Facturación.
- Empleados.

Desventajas:

- Es un software más orientado a la parte económica.

2.4.2. Fly Solutions

Fly Sports es la herramienta especializada en la gestión de clubes deportivos. No sólo se puede gestionar la parte Comercial, Logística y Administrativa gracias a Fly Basic, si no que con la incorporación de 7 módulos se puede gestionar todo el Área Deportiva de cualquier club.
Funcionalidades:

- Masa social: abonados, accionistas, competiciones, accesos, entradas...
- Secretaría Técnica: clubes, ojeadores, representantes, contratos...
- Comunicación: medios, periodistas, redes sociales...
- Club: equipos, jugadores, entrenadores, entrenamientos, base de datos de ejercicios, convocatorias...

Desventajas:

- Precio.

2.5. Selección de la solución

Dado el carácter del club para el que se desarrolla el proyecto, se opta por una aplicación a medida para usar de aquí en adelante y así no depender de software de terceros. A posterior se podría instar al resto de clubes a utilizar el mismo software y crear así una masa social en el sector.

La explotación comercial podría ser de diferentes maneras:

1. Con publicidad en la portada pública.
2. Poniendo límites de equipos y cobrar por equipos extra.

Todo esto si el software se convirtiera en un portal estilo WordPress[18] y los usuarios se hicieran una cuenta y pudieran administrar sus entidades.

2.6. Conclusión

Este proyecto parte de una iniciativa propia del club. Al ser yo mismo el encargado de informática, siempre he ido haciendo modificaciones sobre lo hecho. Ahora se pretende hacer un gestor desde cero y que pueda ser reutilizado por otras entidades que quieran hacerlo.

[18] Software muy extendido de bitácoras

3. Planificación

En este apartado se recoge el plan de proyecto, es decir, el conjunto de actividades que permiten desarrollar, ejecutar y controlar el proyecto. Se utilizarán un seguido de herramientas para facilitar esta tarea, como por ejemplo Planner, una alternativa de código abierto para Microsoft Project.Se incluyen: las tareas,

los puntos de control, los recursos, el calendario, la evaluación de riesgos y el presupuesto del proyecto. Se explican la descomposición del trabajo, indicando cada una de sus fases, con un diagrama donde veremos gráficamente estas fases y las fechas para cada fase.

3.1. Fases y actividades del proyecto

En la tabla 3.1 se especifica la lista de fases del proyecto con su descripción. Entendemos como fase el conjunto de actividades asociadas con el proyecto que se llevan a cabo.

Fase	Descripción
Iniciación.	Se incluyen las actividades definidas del proyecto, asignación y matriculación en el centro universitario.
Planificación.	Se incluye el estudio de viabilidad.
Análisis.	Análisis de requisitos y arquitectura del sistema.
Diseño.	Diseño de la capa de datos, control y la interficie.
Desarrollo.	Fase de desarrollo de la aplicación.
Test y prueba.	Realización de pruebas para asegurar el buen funcionamiento.
Implantación.	Hacer el despliegue en el servidor de producción[19].
Generación de documentos.	Memoria del proyecto.
Finalización.	El director del proyecto firma la aceptación y finalización.
Defensa.	Defensa del proyecto ante la comisión correspondiente.

Tabla 3.1 Fases y actividades.

En la figura 3.1 se puede observar el árbol que describe gráficamente las fases anteriormente descritas. Este árbol es una estructura de descomposición del trabajo: exhaustiva, jerárquica y descendente formada por los entregables a realizar en el proyecto. Esta estructura recibe el nombre de Work Breakdown Structure.

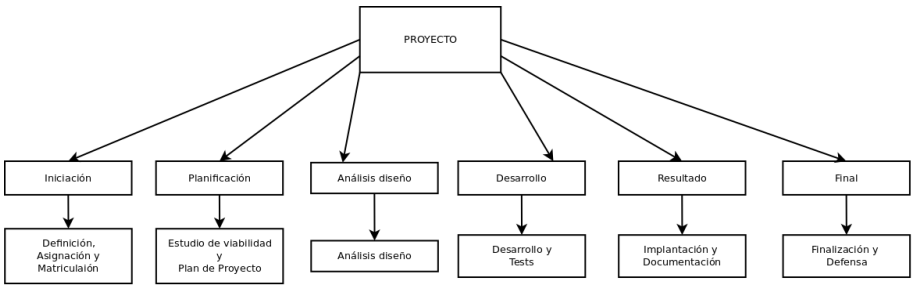

Figura 3.1 Work Breakdown Structure.

3.2. Recursos del proyecto

En la tabla 3.2 se detallan los recursos asignados y se valoran cada uno.

[19] servidor que está funcionando y donde hay que subir las aplicaciones una vez comprobado su funcionamiento para minimizar los perjuicios a los usuarios

Resursos humanos	Valoración
Jefe de proyecto.	100€/h.
Analista.	40€/h.
Programador.	25€/h.
Técnico de pruebas.	18€/h.

Tabla 3.2 Recursos del proyecto

Todo el proyecto se realiza utilizando herramientas de código abierto así que el coste en software será de 0€.

El servidor de Git utilizado para el proyecto se trata de GitHub[20]. Es gratuito para proyectos de código abierto pero, en este caso, se opta por la versión de 6€/mes que permite manejar 5 repositorios privados; una vez finalizado el proyecto se meditará si liberarlo bajo licencia de código abierto.

3.3. Calendario del proyecto

El proyecto se desarrollará desde el 3 de septiembre de 2012 al 29 de agosto de 2013. La dedicación semanal rondará las 10 horas. El total de horas dedicadas al proyecto rondarán las 450 horas.

Todas las fases se desarrollan utilizando un modelo lineal. Cada fase no se iniciará hasta que no se complete la fase anterior.

La fase de generación de documentos se prevé para el final dado que incluirá los documentos elaborados durante el desarrollo del proyecto.

En la siguiente tabla (tabla 3.3) se refleja las fechas de inicio de cada una de las fases.

Fase	Descripción	Fecha
Iniciación.	Matriculación.	03/09/2012
Estudio de viabilidad.	Aprobación.	14/09/2012
Plan del proyecto.	Aprobación.	21/09/2012
Análisis.	Aprobación.	23/10/2012
Diseño.	Aprobación.	5/12/2012
Implementación.	Programación.	28/08/2013
Finalización.	Aprobación.	29/08/2013
Defensa.	Evaluación.	Principios octubre.

Tabla 3.3 Recursos del proyecto.

3.3.1. Calendario de los recursos

Los recursos humanos se utilizarán durante todo el proceso de desarrollo. En este apartado asignaremos a cada recurso (ver tabla 3.2) un periodo en el que serán utilizados.

El **jefe de proyecto** participará durante la fase de iniciación, planificación, generación de documentos, finalización y defensa del proyecto. Definirá los puntos de control.

[20]Servidor remoto para almacenar proyecto cuyo control de versiones es manejado mediante Git.

El **analista** participará en las fases de análisis y diseño, también en los puntos de control del análisis y desarrollo.

El **programador** participará en la fase de desarrollo e implementación. Así, también hará una breve formación a los futuros usuarios de la aplicación.

Para terminar, el **técnico de pruebas** realizará las tareas en la fase de pruebas y las pruebas una vez implementado en un servidor de producción.

3.3.2. Cuadro de tareas del proyecto

En la siguiente figura (Figura 3.2) observamos el cuadro de tareas y subtareas. Se indican los recursos necesarios así como el porcentaje en el que son necesarios. Para cada tarea y subtarea se indican la fecha de inicio y la de fin.

1	Inicio del proyecto, asignación y matrícula	sep 3	sep 6	4d		800	Jefe de proyecto
2	**Planificación**	**sep 7**	**sep 27**	**14d 1h**	**212d 1h**	**1.700**	
2.1	Estudio de viabilidad	sep 7	sep 13	5d		400	Analista
2.2	Aprobación Estudio Viabilidad	sep 14	sep 14	1h		100	Jefe de proyecto
2.3	Plan del proyecto	sep 14	sep 21	5d		400	Analista
2.4	Aprobación Plan del Proyecto	sep 21	sep 27	4d		800	Jefe de proyecto
2.5	Punto de control	sep 27	sep 27	N/D	212d 1h	0	Jefe de proyecto
3	**Análisis de la aplicación**	**sep 27**	**oct 24**	**14d**	**198d 1h**	**1.240**	
3.1	Análisis de requisitos (casos de uso)	sep 27	oct 4	5d		400	Analista
3.2	Análisis de los datos (base de datos)	oct 4	oct 16	3d 1h		280	Analista
3.3	Análisis de la seguridad y legalidad	oct 17	oct 18	1d 1h		120	Analista
3.4	Documentación del análisis	oct 18	oct 23	3d		240	Analista
3.5	Aprobación del análisis	oct 23	oct 24	1d		200	Jefe de proyecto
3.6	Punto de control	oct 24	oct 24	N/D	198d 1h	0	Jefe de proyecto
4	**Diseño de la aplicación**	**oct 24**	**dic 5**	**30d 1h**	**168d**	**2.560**	
4.1	Diseño de la base de datos	oct 24	nov 7	10d		800	Analista
4.2	Diseño modular de la aplicación	nov 7	nov 13	4d 1h		360	Analista
4.3	Diseño de la interficie	nov 14	nov 23	8d		640	Analista
4.4	Diseño de las pruebas	nov 26	nov 29	4d		320	Analista
4.5	Documentación del diseño	nov 30	dic 4	3d		240	Analista
4.6	Aprobación del diseño	dic 5	dic 5	1d		200	Jefe de proyecto
4.7	Punto de control	dic 5	dic 5	N/D	168d	0	Jefe de proyecto
5	**Desarrollo de la aplicación**	**dic 6**	**may 29**	**111d 1h**	**56d 1h**	**5.599**	
5.1	Preparación del entorno de desarrollo	dic 6	dic 10	2d 1h		125	Programador
5.2	Configuración de la base de datos	dic 10	dic 14	4d		224	Analista, Programador
5.3	Desarrollo del back-end	dic 14	may 15	95d		4.750	Programador
5.4	Desarrollo del front-end	may 15	may 29	10d		500	Programador
5.5	Punto de control	may 29	may 29	N/D	56d 1h	0	
6	**Tests y pruebas**	**may 29**	**ago 19**	**48d 1h**	**8d**	**1.828**	
6.1	Pruebas unitarias	may 29	jun 10	8d		288	Técnico de prueba
6.2	Pruebas de integración	jun 10	jun 20	8d		288	Técnico de prueba
6.3	Pruebas de estrés	jun 20	jun 26	4d		144	Técnico de prueba
6.4	Documentación del desarrollo	jun 26	ago 19	28d		1.008	Técnico de prueba
6.5	Aprobación del desarrollo y pruebas	ago 19	ago 19	1h		100	Jefe de proyecto
6.6	Punto de control	ago 19	ago 19	N/D	8d	0	
7	**Implantación**	**ago 20**	**ago 29**	**8d**		**344**	
7.1	Subida al servidor de producción	ago 20	ago 21	2d		100	Programador
7.2	Pruebas reales	ago 22	ago 27	4d		144	Técnico de prueba
7.3	Formación de usuarios	ago 28	ago 29	2d		100	Programador
8	Cierre del proyecto	ago 29	ago 29	N/D		0	Jefe de proyecto

Figura 3.2 Cuadro de tareas.

3.3.3. Diagrama de Gantt

En la siguiente figura (Figura 3.3) se muestra el diagrama de Gantt y el camino crítico del proyecto.

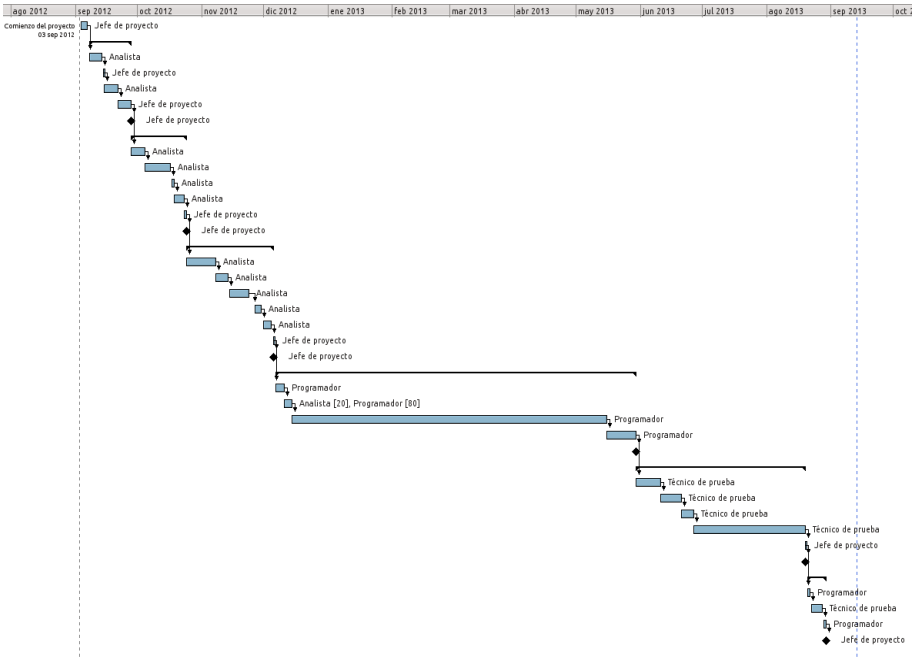

Figura 3.3 Diagrama de Gantt.

3.4. Coste del proyecto

Una vez valorados los costes de los recursos y habiendo planificado el proyecto, podemos calcular el coste total del proyecto.

Se utilizarán los precios visto en la tabla 3.2 para la estimación total del coste. Aún así, hay que tener en cuenta que será el proyectista el encargado de ejercer los roles tanto de analista, programador como técnico de pruebas.

El coste total se puede ver en la siguiente tabla (Tabla 3.4).

Fase	Coste
Iniciación	800€
Planificación	1700€
Análisis	1240€
Diseño	2560€
Desarrollo	5599€
Tests y documentación	1828€
Implantación	344€
TOTAL	**14071€**

Tabla 3.4 Coste del proyecto.

3.5. Evaluación de riesgos

En este apartado se describen y evalúan los riesgos. Se tiene en cuenta el impacto que estos pueden causar y se valoran posibles soluciones. Se tienen en cuenta tanto los riesgos inherentes al problema como los derivados de la solución propuesta.

3.5.1. Lista de riesgos

1. **Planificación optimista**: provocado por una mala estimación de la duración. El proyecto se terminaría con fecha posterior y esto provocaría un aumento de recursos.

2. **Falta de tarea necesaria**: tendría su origen en el plan de proyecto. Los objetivos no serían alcanzados.

3. **Cambio de requisitos**: producido en el estudio de viabilidad o análisis. Provocaría un retraso en el desarrollo.

4. **Herramientas de desarrollo inadecuadas**: en la fase de desarrollo. Retraso en la finalización, proyecto de menor calidad...

5. **Dificultad para acceder a los Stakeholders**: en el estudio de viabilidad, en la fase de análisis, pruebas o formación. Provocaría falta de requisitos, atrasos e insatisfacciones de los usuarios.

6. **Fase de test incorrecta**: durante el desarrollo y los tests. Falta de calidad, deficiencias, insatisfacciones y pérdida económica.

7. **Incumplimiento de alguna norma o legislación**: puede darse en cualquier fase. No se cumplen los objetivos; posibles repercusiones legales en un futuro.

8. **Abandono del proyecto**: puede ocurrir en cualquier fase. Puede provocar pérdidas económicas y frustración personal.

9. **Dificultades en la implementación**: durante la fase de desarrollo. Provocaría que el proyecto no finalizase, se retrasase o finalizara incorrectamente.

3.5.2. Catalogación de los riesgos

En la siguiente tabla (Tabla 3.5) se hace una relación de los riesgos de la lista anterior con la probabilidad que tiene de que sucedan y, en caso que sucedieran, el impacto que podría tener.

Los niveles de probabilidad serán: alto, media o bajo.

Los niveles de impacto serán: catastróficos, crítico o marginal.

En el caso de ser catastróficos será de difícil recuperación el proyecto. En el caso de ser marginal no influiría demasiado en el desarrollo.

Riesgo	Probabilidad	Impacto
R1	Alta	Crítico
R2	Alta	Crítico
R3	Alta	Marginal
R4	Baja	Crítico
R5	Media	Crítico
R6	Alta	Crítico
R7	Media	Crítico
R8	Baja	Catastrófico
R9	Baja	Crítico

Tabla 3.5 Catalogación de los riesgos.

3.5.3. Plan de contingencia

En la siguiente tabla (Tabla 3.6) se explican las posibles soluciones para cada posible riesgo.

Riesgo	Solución
R1	Aplazar alguna funcionalidad, afrontar posibles pérdidas.
R2	Revisar el plan de proyecto y modificar la planificación.
R3	Revisar las funcionalidades y reevaluar.
R4	Prever herramientas alternativas y mejorar la calidad.
R5	Fijar un calendario de reuniones y mejorar el contacto.
R6	Diseñar los tests con antelación. Realizar tests automáticos.
R7	Revisar las normas y legislación vigente. Consultar a un experto y afrontar posibles repercusiones legales.
R8	Sin solución.
R9	Repasar conceptos. Buscar documentación de apoyo.

Tabla 3.6 Plan de contingencia.

4. Fundamentos teóricos

Con la finalidad de desarrollar correctamente el proyecto, se precisan algunos fundamentos teóricos relacionados con el diseño y desarrollo de la aplicación a realizar.

4.1. Aplicaciones web

4.1.1. ¿Qué es una aplicación web?

Una aplicación web es aquella aplicación en la que los usuarios pueden acceder a través de un navegador web.

Las aplicaciones web son populares debido a la practicidad que poseen; solo se precisa de un navegador, independientemente del sistema operativo o la tecnología del hardware. Este tipo de tecnologías facilitan la distribución del mismo; solo hay que instalar la aplicación en el servidor y todos los usuarios tienen la versión que hemos instalado disponible. No cabe decir que las actualizaciones son igual de sencillas en comparación con otro tipo de software en el que deberíamos ir, uno a uno, actualizando cada PC.

El funcionamiento básico de una aplicación web, y en el que se basa su potencia y sencillez, es en la pareja: petición - respuesta. En una aplicación web todo es una petición que realiza el usuario (solicitar una web, enviar formularios...) y una respuesta que ofrece el servidor (index.php, página no encontrada, error en los datos...). Con esta sencilla premisa se pueden hacer aplicaciones muy complejas.

4.1.2. Estructura de una aplicación web

Aunque existen muchas variaciones posibles, una aplicación web está normalmente estructurada como una aplicación de tres capas. En su forma más común el navegador ofrece la primera capa, un motor capaz de usar alguna tecnología web dinámica[21] que constituye la capa intermedia y una base de datos constituye la tercera y última capa.

El navegador web manda peticiones a la capa intermedia que ofrece servicios valiéndose de consultas y actualizaciones a la base de datos y a su vez proporciona una interfaz de usuario.

Esta estructura básica la podemos ver en la siguiente figura (Figura 4.1):

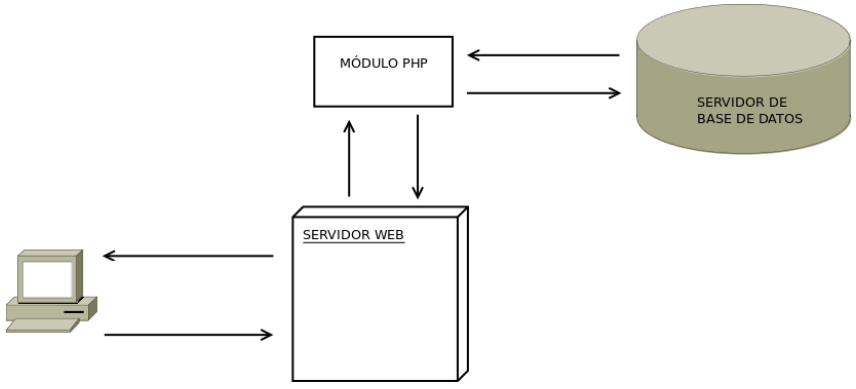

Figura 4.1 Estructura básica aplicación web.

[21] ejemplo: PHP, Java Servlets o ASP, ASP.NET, CGI, ColdFusion, embPerl, Python (programming language) o Ruby on Rails

En la figura podemos observar como el usuario actúa contra un servidor web. Éste, a su vez, hace lo propio contra su módulo de PHP (lenguaje en el que se basa nuestra aplicación) y ejecuta consultas contra una base de datos. El módulo que ejecuta el código PHP forma parte del propio servidor web; se especifica de forma separada porque no forma parte del núcleo.

4.1.3. Lenguaje HTML

HTML, siglas de HyperText Markup Language (lenguaje de marcado hipertextual), se trata del lenguaje de marcado[22] más extendido que existe hoy en día. Se utiliza para describir y traducir la estructura e información en forma de texto, así como complementar el texto con objetos como por ejemplo: las imágenes. El HTML se escribe en forma de "etiquetas", rodeadas por corchetes angulares (<,>). HTML también puede describir, hasta un cierto punto, la apariencia de un documento; y puede incluir o hacer referencia a un tipo de programa llamado script, el cual puede afectar al comportamiento de navegadores web y otros procesadores de HTML.

HTML consta de varios componentes vitales, entre ellos los elementos y sus atributos, tipos de datos y la declaración de tipo de documento.

Elementos

Los elementos son estructura básica de HTML. Estos tienen dos propiedades básicas: atributos y contenido. Cada atributo y contenido tiene ciertas restricciones para que se considere válido el documento HTML. Un elemento, generalmente, posee una etiqueta de inicio (<nombre-del-elemento>) y una etiqueta de cierre (</nombre-del-elemento>). Los atributos están contenidos en la etiqueta de inicio y el contenido está ubicado entre las dos etiquetas.

Código 4.1 Estructura elemento HTML.

```
<elemento atributo="valor atributo"> contenido del elemento
    </elemento>
```

Aquí podemos ver la estructura que hemos descrito antes. Ahora veamos un ejemplo real del lenguaje:

Código 4.2 Ejemplo real HTML.

```
<p class="documento"> Aqui podemos ver la estructura que hemos
    descrito antes. </p>
```

En este caso tenemos un elemento **p** que nos indica que lo que viene a continuación es un parágrafo. Utilizamos el atributo **class** para darle una clase y así, posteriormente, poderle asignar estilo o tratar el elemento por medio de scripting. Y luego tenemos el contenido, todo lo que tenemos entre el inicio y el final de las etiquetas.

[22] Un lenguaje de marcado o lenguaje de marcas es una forma de codificar un documento que, junto con el texto, incorpora etiquetas o marcas que contienen información adicional acerca de la estructura del texto o su presentación.

4.1.4. Hojas de estilo en cascada o CSS

Conocidas como **CSS**, sus siglas en inglés (**Cascading Style Sheests**), se trata de un lenguaje para describir la presentación semántica (aspecto y formato) de un documento escrito en lenguaje de marcas. Su aplicación más común es dar estilo a páginas webs, aunque no es la única. De todas maneras nos centraremos únicamente en este aspecto que es el que nos concierne.

Sintaxis

CSS posee una sintaxis muy sencilla, que usa unas cuantas palabras tomadas del inglés para especificar los nombres de sus selectores, propiedades y atributos.

Los selectores marcarán qué elementos se verán afectados por cada bloque de estilo que les siga, pudiendo afectar a uno o varios elementos a la vez, en función de su tipo, nombre (name), ID, clase (class), posición dentro del DOM[23],etcétera.

Abajo puede verse un ejemplo de una parte de una hola de estilos:

Código 4.3 Estructura CSS.

```
selector [, selector2, ...] [:pseudo-class][::pseudo-element] {
  propiedad: valor;
  [propiedad2: valor2;
  ...]
}

/* comentarios */
```

Aplicado a nuestro ejemplo anterior, ahora podríamos definir un estilo para nuestro elemento **p** que posee la clase **documento**:

Código 4.4 Ejemplo CSS básico.

```
p.documento{
  font-size: 14px;
  font-weight: bold;
}
```

En este caso le estamos especificando que el contenido del elemento **p** con clase **documento** debe tener un tamaño de fuente de 14 píxeles y este texto debe estar en negrita (bold).

4.1.5. less CSS

Se trata de una ampliación a las hojas de estilo CSS que intenta solventar varias deficiencias que estas poseen. Estas deficiencias son: uso de variables, funciones (o mixins) y operaciones aritméticas.

[23]o Document Object Model. Es esencialmente una interfaz de programación de aplicaciones (API) que proporciona un conjunto estándar de objetos para representar documentos HTML y XML

Uso de variables

Una de las especificaciones que se esperaban en CSS con su versión 3 era la inclusión de variables para facilitar el mantenimientos y la modificación de estilos. Para entenderlo plantearemos un ejemplo basándonos en nuestra clase documento.

Anteriormente hemos aplicado un tamaño de fuente de 14 píxeles al elemento **p**. Ahora supongamos que queremos crear otra clase y la llamaremos **documento2**, a la que también queremos aplicarle el mismo tamaño de fuente. Podríamos volver a definir el valor font-size en 14px y así ya lo tendríamos. Así nos quedaría nuestra hoja de estilos:

Código 4.5 CSS sin variables.

```
p.documento{
  font-size: 14px;
  font-weight: bold;
}

p.documento2{
  font-size: 14px;
}
```

Si ahora, por cualquier razón, nuestro diseñador nos solicitara cambiar el tamaño de 14 a 12 píxeles deberíamos hacer el cambio en las dos clases. Ahora imaginemos un proyecto real con varias hojas de estilo y cientos o miles de líneas de código; ¡Nos podríamos volver locos buscando!

Para facilitar estos cambios less nos introduce el uso de variables en nuestras hojas de estilo. Pongamos el ejemplo anterior como un documento less:

Código 4.6 Uso de variables.

```
@font14px: 14px;

p.documento{
  font-size: @font14px;
  font-weight: bold;
}

p.documento2{
  font-size: @font14px;
}
```

Ahora, en caso de solicitar un cambio de tamaño solo deberíamos hacer el cambio en la variable. Una vez hecho procederíamos a compilar el fichero **less** en **CSS** y así poder incorporar nuestra hoja a nuestra web.

Mixins

Podríamos definirlos como las "funciones" de less. Se puede usar para reutilizar una clase ya definida y no tener que repetir una y otra vez lo mismo:

Código 4.7 Uso de mixin básico.

```
.documento_base{
  font-size: 14px;
}

p.documento{
  .documento_base;
  font-weight: bold;
}

p.documento2{
  .documento_base;
}
```

En este caso extendemos el uso de la clase **documento_base** a las otras dos clases para que partan de unas especificaciones básicas que sabemos que van a ser siempre estas. Luego podemos añadir más atributos, como vemos en nuestra clase **documento**. También podemos usar mixins y pasarle variables para crear unas especificaciones en las que poder modificar ciertos valores:

Código 4.8 Uso de mixin con parámetros.

```
.boton(@fondo: #ddd, @texto: #000){
  border-radius:4px;
  border:solid 1px #aaa;
  background:@fondo;
  color:@texto;
}

.boton_verde{
  .boton(#0f0, #fff);
}
```

En este caso tenemos un mixin que nos define una clase **boton** con ciertos atributos pasados por parámetros. En la clase **boton_verde** llamamos a nuestro mixin antes definido y le pasamos los parámetros: @fondo es #0F0 (verde en RGB[24]) y @texto es #FFF (blanco en RGB).

Operaciones aritméticas

Como hemos adelantado, less nos permite la inclusión de operaciones aritméticas. Para entender el concepto iremos con un ejemplo básico basado en nuestras clases ya conocidas. Imaginemos que ahora queremos que nuestra clase **documento2** siempre tenga 2 píxeles menos que nuestra clase **documento**. Less nos permite hacer esto de forma muy sencilla.

[24] RGB (en Inglés Red, Green, Blue, en Español rojo, verde y azul) es la composición del color en términos de la intensidad de los colores primarios de la luz

Código 4.9 Operaciones aritméticas en less.

```
@font14px: 14px;
@fontMenos: (@font14px - 2);

p.documento{
  font-size: @font14px;
  font-weight: bold;
}

p.documento2{
  font-size: @fontMenos;
}
```

Como vemos, tenemos nuestra variable @fontMenos que siempre tendrás 2 píxeles menos que nuestra variable @font14px.

4.1.6. JavaScript

Se trata de un lenguaje de programación interpretado. Se define como orientado a objetos[25], basado en prototipos[26], imperativo[27], débilmente tipado y dinámico.

Se utiliza, principalmente, en su forma del lado del cliente (client-side); implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. También existe una forma de JavaScript del lado del servidor (Server-side JavaScript o SSJS). Su uso en aplicaciones externas a la web, por ejemplo en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets) es también significativo.

JavaScript se diseñó con una sintaxis similar al C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo, Java y JavaScript no están relacionados: tienen semánticas y propósitos diferentes.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

Tradicionalmente se venía utilizando en páginas web HTML para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor. JavaScript se interpreta en el agente de usuario, al mismo tiempo que las sentencias van descargándose junto con el código HTML.

A continuación tenemos un ejemplo de código.

[25] es un paradigma de programación que usa los objetos en sus interacciones, para diseñar aplicaciones y programas informáticos

[26] es un estilo de programación orientada a objetos en el cual los objetos no son creados mediante la instanciación de clases sino mediante la clonación de otros objetos o mediante la escritura de código por parte del programador

[27] es un paradigma de programación que describe la programación en términos del estado del programa y sentencias que cambian dicho estado. Los programas imperativos son un conjunto de instrucciones que le indican al computador cómo realizar una tarea

Código 4.10 Ejemplo función factorial.

```
function factorial(n) {
  if (n === 0) {
      return 1;
  }
  return n * factorial(n - 1);
}
```

4.1.7. jQuery

Se trata de una biblioteca de JavaScript, de las más utilizadas hoy en día, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.

jQuery es software libre y de código abierto, posee un doble licenciamiento bajo la Licencia MIT y la Licencia Pública General de GNU v2, permitiendo su uso en proyectos libres y privativos. Al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio.

La función $()

La forma de interactuar con la página es mediante la función $(), un alias de jQuery(), que recibe como parámetro una expresión CSS o el nombre de una etiqueta HTML y devuelve todos los nodos (elementos) que concuerden con la expresión. Esta expresión es denominada selector en la terminología de jQuery.

Código 4.11 Ejemplo de uso de jQuery.

```
// Se elimina el estilo y se aplica uno nuevo a todos los nodos con
    class="activo"
$(".activo").removeClass("activo").addClass("inactivo");
```

Como puede verse en el ejemplo, la interacción con los elementos de la página web se vuelve extremadamente sencilla; pudiendo, con una simple línea de código, eliminar una clase y añadir otra.

Una de las grandes ventajas que aporta jQuery en la interacción con el usuario es la facilidad de definir la captura de eventos que se producen.

Código 4.12 Ejemplo de uso de jQuery.

```
$(".activo").click(function(){
  alert('Has hecho click!');
});
```

Como podemos observar es muy fácil lanzar una función en el momento que el usuario hace click en cualquier elemento que posea la clase **activo**.

CKEditor

Se trata de un editor WYSIWYG[28] escrito en JavaScript. Cambia un elemento **textarea** de HTML en un completo editor con múltiples opciones tales como: cursiva, negrita, tamaño de fuente...

Es utilizado en numerosos proyectos, el más conocido es WordPress.

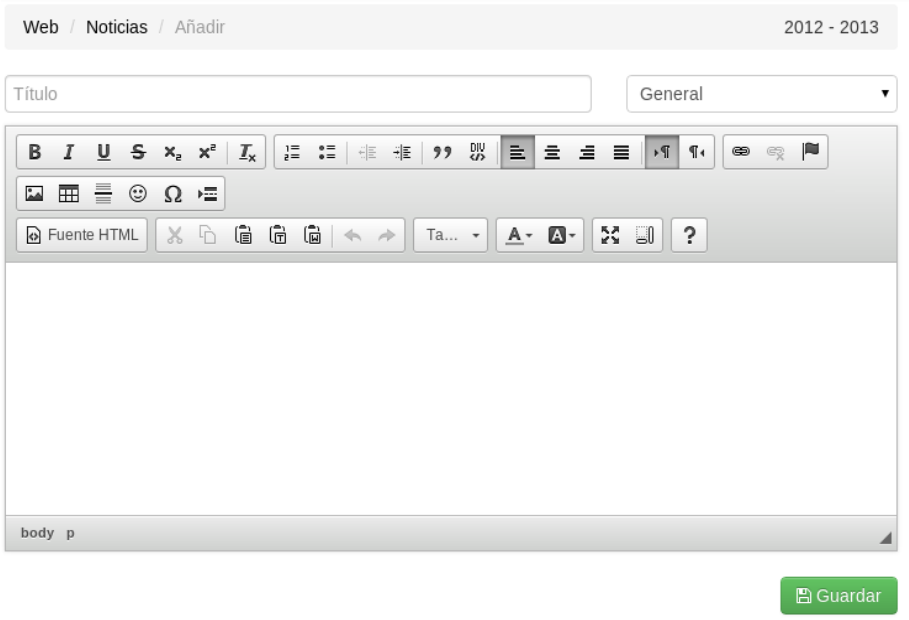

Figura 4.2 CKEditor dentro del proyecto.

4.1.8. Boostrap

Según los mantenedores del código: "Bootstrap is a sleek, intuitive, and powerful front-end framework for faster and easier web development".

Básicamente se trata de un framework para el rápido desarrollo de front-ends[29] que hace uso de clases predefinidas de CSS y plugins[30] JavaScript.

Gracias a ello se puede tener una interficie limpia y amigable sin mucho esfuerzo para así poderse centrar en la funcionalidad y el desarrollo de la lógica.

Tanto para la zona pública (o front-end) como para la zona de administración (o back-end) del proyecto se utiliza la versión 2.3.2.

4.1.9. PHP

Hasta ahora hemos visto las tecnologías que llegan al navegador del usuario y que este interpreta para la interacción; que ocurre tras la petición del usuario al servidor web. Pero antes de recibir, es en el servidor dónde ocurre la "magia"

[28] What You See Is What You Get o lo que escribes es lo que obtienes
[29] parte pública de una aplicación web
[30] programa externo que añade funcionalidades a otro programa o aplicación

de una aplicación web: se realizan los cálculos y peticiones a la base de datos que dan como resultado la página que estamos viendo. Y de esto, en nuestro proyecto, se encarga uno de los lenguajes más conocidos: PHP.

PHP es un lenguaje de programación de uso general de código del lado del servidor. Fue uno de los primeros lenguajes de programación del lado del servidor que se podían incorporar directamente en el documento HTML en lugar de llamar a un archivo externo que procese los datos. El código es interpretado por un servidor web con un módulo de procesador de PHP que genera la página Web resultante. PHP ha evolucionado por lo que ahora incluye también una interfaz de línea de comandos que puede ser usada en aplicaciones gráficas independientes. PHP puede ser usado en la mayoría de los servidores web al igual que en casi todos los sistemas operativos y plataformas sin ningún costo.

PHP fue creado originalmente por Rasmus Lerdorf en 1995. Actualmente el lenguaje sigue siendo desarrollado con nuevas funciones por el grupo PHP. Este lenguaje forma parte del software libre publicado bajo la licencia PHP que es incompatible con la Licencia Pública General de GNU debido a las restricciones del uso del término PHP.

Un uso típico de PHP lo podemos ver a continuación.

Código 4.13 Ejemplo de uso de PHP.

```
<HTML>
...
<p>
  <?php
    echo "hola mundo";
  ?>
</p>
...
</HTML>
```

Como podemos ver, y hemos dicho antes, se puede intercalar código de PHP en nuestros ficheros HTML. Esto se ejecuta en el servidor y se cambia por el resultado; y una vez hecho se devuelve al usuario que no necesita saber qué ni cómo llega el código a su navegador.El código de antes devolvería el siguiente código al usuario:

Código 4.14 Ejemplo de uso de PHP.

```
<HTML>
...
<p>
hola mundo
</p>
...
</HTML>
```

En este caso se trata de un simple **echo**. Lo único que hace es sacar por pantalla lo que tiene entre comillas (“). El usuario no verá las etiquetas de apertura y cierre ni la instrucción que ha ejecutado el servidor para obtener los datos.

Una de las funciones más llamativas y las que proveen de una gran potencia

a PHP es la posibilidad de acceder a una base de datos externa, recuperar los datos que han sido solicitados por el usuario y devolvérselos en formato HTML, JSON,texto plano, etcétera.

4.2. Bases de datos

Una base de datos, o banco de datos, es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. En este sentido, una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta. Actualmente, y debido al desarrollo tecnológico de campos como la informática y la electrónica, la mayoría de las bases de datos están en formato digital, y por ende se ha desarrollado y se ofrece un amplio rango de soluciones al problema del almacenamiento de datos.

Existen programas denominados sistemas gestores de bases de datos, abreviado SGBD, que permiten almacenar y posteriormente acceder a los datos de forma rápida y estructurada. Las propiedades de estos SGBD, así como su utilización y administración, se estudian dentro del ámbito de la informática.

Las aplicaciones más usuales son para la gestión de empresas e instituciones públicas. También son ampliamente utilizadas en entornos científicos con el objeto de almacenar la información experimental.

Existen diferentes tipos de bases de datos. Aquí trataremos solo las bases de datos relacionales; son las más utilizadas actualmente.

En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia. Esto tiene la considerable ventaja de que es más fácil de entender y de utilizar para un usuario esporádico de la base de datos. La información puede ser recuperada o almacenada mediante "consultas" que ofrecen una amplia flexibilidad y poder para administrar la información.

El lenguaje más habitual para construir las consultas a bases de datos relacionales es SQL, Structured Query Language o Lenguaje Estructurado de Consultas, un estándar implementado por los principales motores o sistemas de gestión de bases de datos relacionales.

4.2.1. MySQL

Dentro de las bases de datos relacionales nos vamos a centrar en MySQL,que es la que usan el proyecto.

MySQL es un sistema de gestión de bases de datos relacional, multihilo y multiusuario con millones de instalaciones. MySQL AB -desde enero de 2008 una subsidiaria de Sun Microsystems y ésta, a su vez, de Oracle Corporation desde abril de 2009- desarrolla MySQL como software libre en un esquema de licenciamiento dual.

Por un lado se ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia, pero para aquellas empresas que quieran incorporarlo en productos privativos deben comprar a la empresa una licencia específica que les permita este uso.

Podríamos decir que casi cualquier hosting mundial posee este tipo de base de datos, es por ello que es de lo más utilizado para pequeños y medianos proyectos; e incluso grandes proyectos también hacen uso, como Facebook.

La mayoría de lenguajes de programación en la parte del servidor poseen librerías para interactuar con bases de datos MySQL. Esto se explica por lo anteriormente mencionado; cualquiera que precie su producto ha de permitir interoperatividad con este tipo de base de datos.

4.2.2. Entidad - Relación

En el proyecto se utiliza un tipo de base de datos relacional: MySQL. Esto significa que tiene unos datos almacenados en estructuras (**entidad**) y estos se pueden relacionar (no es necesario) con otras entidades, (**Relación**).

Para entender este concepto se representan dos entidades de la base de datos utilizada en el proyecto. La forma habitual de representar una entidad gráficamente es en forma de tabla. Así que podemos simplificar diciendo que tenemos dos "tablas":

Equipos	
Nombre	**Tipo**
id	INT
nombre	Varchar
diminutivo	Varchar
dia	INT
hora	TIME
id_liga	INT
id_temoporada	INT
nuestro	TINYINT
foto	VARCHAR

Ligas	
Nombre	**Tipo**
id	INT
nombre	Varchar
grupo	INT
id_categoria	INT
id_temporada	INT
puntos_ganar	INT
puntos_empatar	INT
puntos_perder	INT
mostrar	TINYINT
orden	TINYINT

Tabla 4.1 Tablas de Equipos y Ligas.

Tal y como observamos en las tablas (Tablas 4.1), existe una relación entre la entidad Equipos y la entidad Ligas. Un equipo pertenece a una liga. La forma en que los relacionamos es añadiendo un campo en la estructura de los equipos que hace referencia a la liga a la que pertenecen: **id_liga**.

4.2.3. Configuración de una entidad

Como se ve en las tablas (Tablas 4.1), existe un campo que nos indica el tipo de dato para cada fila. Cada fila de la tabla es un dato que se almacenará dentro de la entidad. Dicho dato tiene una serie de atributos que hay que especificar:

1. **Tipo de dato**: para optimizar la forma de almacenar los datos se nos solicita qué tipo de dato va a almacenar. Así podemos tener diferentes tipos como: VARCHAR, INT, LONGINT, TINYINT, DATETIME... Tipos bastante genéricos y explicativos por su propio nombre.

2. **Índice**: debemos indicar si se trata de un índice. En cuyo caso el Sistema Gestor de Bases de Datos (en nuestro caso MySQL) ordenará por dicho elemento para optimizar las búsquedas.

3. **Única**: indicamos que no puede repetirse el valor dentro de la misma entidad para ese elemento. Por ejemplo, el campo e-mail para un registro de usuarios nos interesa que no esté repetido.

4. **Clave primaria**: o primary key (PK) en inglés. Es obligatoria y se trata del elemento (o unión de elementos) que identifica de forma única a la totalidad de una fila dentro de la entidad.

5. **Otros**: podemos indicar también que el elemento pueda o no ser nulo, que la PK se auto-incremente (típico ejemplo de generación de ID), el esquema predeterminado de los datos (ejemplo 0000-00-00 para fechas)...

4.3. MVC o Modelo-Vista-Controlador

El proyecto está desarrollado bajo la premisa de un sistema Modelo-Vista-Controlador (MVC en adelante). Se trata de un patrón de arquitectura de software que separa los datos y la lógica de una aplicación de la interfaz de usuario. Para ello, MVC propone la construcción de tres componentes: el **modelo**, la **vista** y el **controlador**. Este patrón de diseño se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

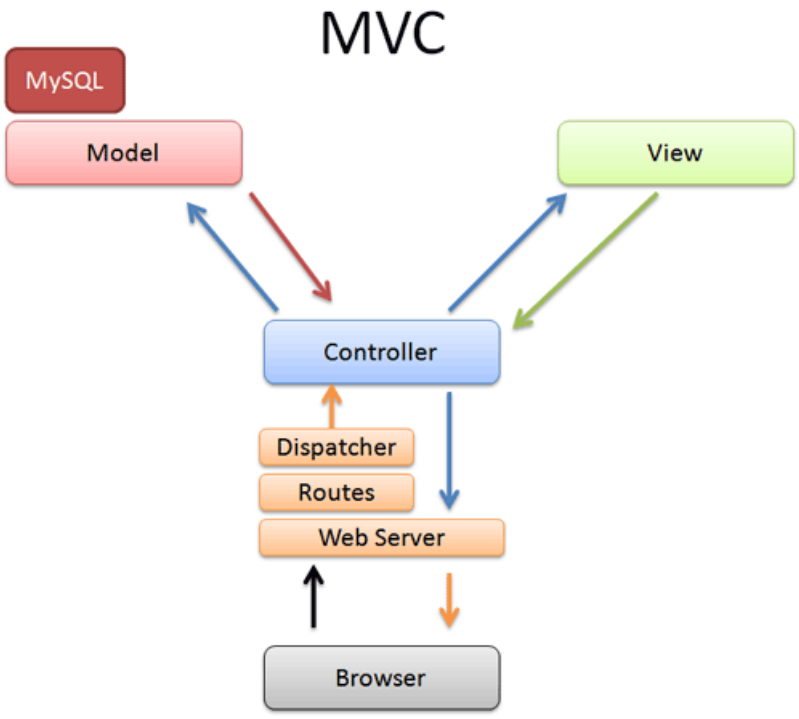

Figura 4.3 Esquema de funcionamiento MVC.

4.3.1. Controlador (Controller)

Es el centro de este tipo de patrón. Recibe las peticiones del usuario, accede a los modelos para pedir datos y renderiza las vistas para devolver a los usuarios. **Todas** las peticiones pasan por un único fichero que es el que se encarga de "controlar".

El flujo básico de una petición sería:

1. El usuario envía una petición al Controlador.
2. El Controlador solicita al Modelo los datos y este se los devuelve.
3. El Controlador solicita la Vista y la renderiza.
4. El Controlador devuelve la Vista al usuario

Como todo ha de pasar por el Controlador, esto nos confiere un mayor control de la aplicación.

4.3.2. Modelo (Model)

Son las representaciones de los datos. Pueden estar almacenados en una base de datos de tipo MySQL o sencillamente pueden ser objetos prefijados con datos constantes. Lo mejor de este sistema es que eso al resto de componentes no le importa; el Controlador solicitará unos datos pero éste no sabe en qué formato están guardados, eso es cosa del Modelo. Así, si en un futuro se precisara cambiar la forma de almacenar los datos de MySQL a MSSQL[31] solo habría que cambiar los Modelos, el resto de la aplicación quedaría igual.

4.3.3. Vista (View)

Se trata de la representación puramente visual. Una vez el Controlador tiene los datos que le ha facilitado el Modelo, renderiza la Vista correspondiente. Es aquí donde escribimos nuestro código HTML, CSS y JavaScript; dado que son los lenguajes utilizados para la parte visual de nuestra aplicación.

Esto nos confiere un control para el mantenimiento muy bueno, indiferente del resto de componentes. Si se precisara cambiar la forma en que la aplicación muestra la información (el diseño gráfico), solo habría que modificar las Vistas.

Twig

Se trata de un motor de plantillas para PHP y es el que usa el proyecto para las vistas. Sus características principales son:

1. **Rápido**: compila las plantillas hasta código PHP regular optimizado. El costo general en comparación con código PHP se ha reducido al mínimo.
2. **Seguro**: tiene un modo de recinto de seguridad para evaluar el código de la plantilla que no es confiable. Esto te permite utilizar Twig como un lenguaje de plantillas para aplicaciones donde los usuarios pueden modificar el diseño de la plantilla.

[31] o Microsoft SQL, es el servidor de bases de datos relacionales de Microsoft

3. **Flexible**: Twig es alimentado por flexibles analizadores léxico y sintáctico. Esto permite al desarrollador definir sus propias etiquetas y filtros personalizados.

¿Por qué usar Twig?

El razonamiento principal es separar la lógica de la parte visual de la aplicación.

Otro de los aspectos que mejoran es la legibilidad del código.

Por último, es bastante más sencillo aprender Twig que PHP para un programador de interfaz de usuario. No es necesario aprender un lenguaje tan completo para beneficiarse de las características del mismo.

5. Herramientas utilizadas

Para un buen desarrollo es imprescindible saber elegir las herramientas idóneas al alcance de las que podemos disponer. Aquí se describen las herramientas utilizadas durante el desarrollo del proyecto.

5.1. LAMP o Linux-Apache-MySQL-PHP

El desarrollo será llevado desde un sistema operativo GNU/Linux. Se instalarán, mediante un gestor de paquetes[32], los programas necesarios:

5.1.1. Apache

Página oficial: http://httpd.apache.org/

Servidor web muy extendido hoy en día. Una de sus ventajas es la configuración de virtual hosts[33], así se pueden tener bastantes aplicaciones distintas con el mismo servidor.

Para empezar el desarrollo se necesita configurar un virtual host que nos servirá para albergar nuestra aplicación y así simular que la tenemos en internet.

Código 5.1 Configuración de un VirtualHost.

```
<VirtualHost *:80>
        DocumentRoot /var/www/new.cfs/cfs
        ServerName admin.cfssantboi.com
    <directory "/var/www/new.cfs/cfs">
        Options Indexes FollowSymLinks
        AllowOverride all
        Order Deny,Allow
        Deny from all
        Allow from 127.0.0.1
    </directory>
        ErrorLog "/var/log/apache2/admin.cfssantboi.com-error.log"
        CustomLog "/var/log/apache2/admin.cfssantboi.com-access.log"
            common
</VirtualHost>
```

En este caso estamos indicando que la petición puede venir de cualquiera y entrará por el puerto 80 (*:80). Solo contestaremos desde este host virtual si el usuario solicita la url http://admin.cfssantboi.com y en ese caso le mostraremos el contenido del directorio local (/var/www/new.cfs/cfs). En el apartado **directory** restringimos el acceso a todo aquel que no sea la misma máquina (Allow from 127.0.0.1) así conseguimos el nivel de seguridad que nadie (ya sea desde la misma red o desde internet, en caso de tener habilitado el puerto 80 para acceder desde fuera) pueda acceder si no somos nosotros mismos. Por último configuramos dos ficheros de log[34] específicos para así encontrar los errores más fácilmente.

Para terminar, hemos de indicar al sistema operativo que en lugar de ir a buscar http://admin.cfssantboi.com a su servidor DNS[35] se dirija directamente a la IP 127.0.0.1, que somos nosotros. Esto, en un sistema ope-

[32]Gestor de aplicaciones oficiales del sistema operativo. Se conecta a servidores oficiales para descargar programas.

[33]Simula una instalación nueva del servidor web acotando a la aplicación web a no salir de ella.

[34]ficheros en los que se almacena el comportamiento y los errores que se producen en la ejecución

[35]Domain Name Server o Servidor de Nombre de Dominio son los servidores encargados de traducir un dominio a la IP de destino para así poder realizar la comunicación.

rativo GNU/Linux, se hace editando el fichero **hosts**[36], que se encuentra en `/etc/hosts`, y añadiendo la siguiente línea:

Código 5.2 Línea en el fichero hosts.

```
127.0.0.1    admin.cfssantboi.com
```

5.1.2. MySQL

Página oficial: http://www.mysql.com/

Ya se ha explicado en Fundamentos teóricos de qué se trata MySQL, así que en este caso solo añadiremos que su instalación y configuración es bastante sencilla. Desde el mismo Gestor de aplicaciones se nos preguntará por la contraseña de administrador para así poder añadir bases de datos. Para ello utilizaremos una aplicación web llamada phpMyAdmin que explicaremos más adelante.

5.1.3. PHP

Página oficial: http://php.net/

Junto al servidor web hemos de elegir también instalar PHP y que así sepa dónde ha de enviar ese código cuando se lo encuentre; tal y como se explica en Fundamento teóricos.

5.2. phpMyAdmin

Página oficial: http://www.phpmyadmin.net/

Se trata de una aplicación escrita en PHP con la intención de manejar la administración de MySQL a través de un navegador web. Sus características principales son:

1. Crear y eliminar bases de datos.
2. Crear, eliminar y alterar tablas (entidades).
3. Borrar, editar y añadir campos.
4. Ejecutar cualquier sentencia SQL.
5. Administrar claves en campos.
6. Administrar privilegios.
7. Exportar datos en varios formatos.
8. Disponible en 62 idiomas.

[36] Este fichero es el primero que mira el sistema operativo antes de consultar con el servidor DNS; si encuentra una entrada que concuerde con lo que busca, resuelve el dominio directamente.

Se encuentra bajo licencia GPL, así que cualquiera puede descargarlo y hacer uso. Una vez configurado y habiéndose autenticado correctamente como administrador, la interficie de usuario es bastante clarificadora. Podemos verlo en la siguiente imagen (Figura 5.1).

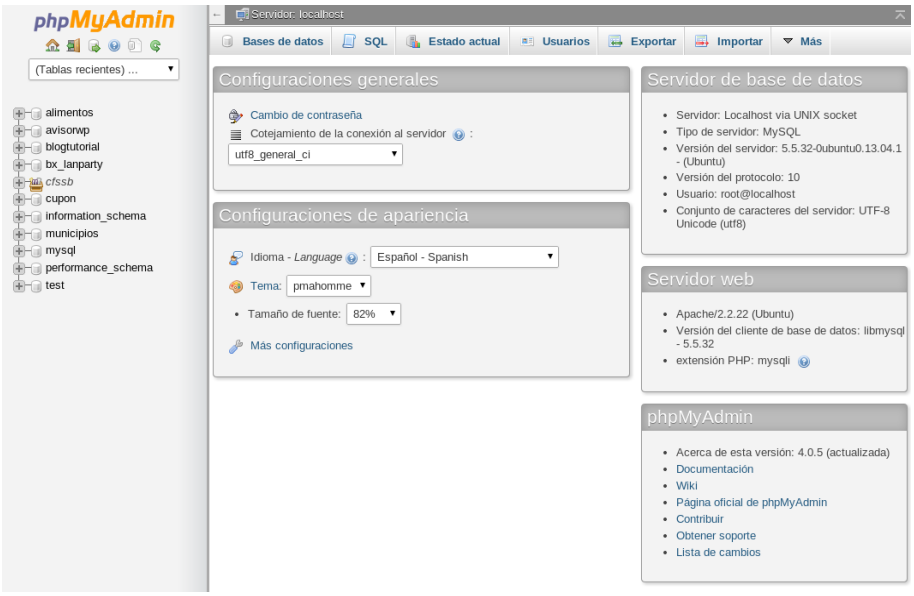

Figura 5.1 Pantalla principal phpMyAdmin.

5.3. NetBeans

Página oficial: https://netbeans.org/

Se trata de un IDE (de las siglas en inglés Integrated Development Enviroment). Básicamente es un programa compuesto por un conjunto de herramientas que ayudan en el proceso del desarrollo. NetBeans fue originalmente pensado para programar con el lenguaje JAVA, pero soporta bastantes más lenguajes.

Como cualquier IDE que se precie hoy en día posee resaltado de sintaxis; esto significa que colorea palabras clave del lenguaje para su mejor legibilidad.

La elección del IDE es bastante personal. Hoy en día el aspecto visual de los IDE se parecen bastante (Figura 5.2).

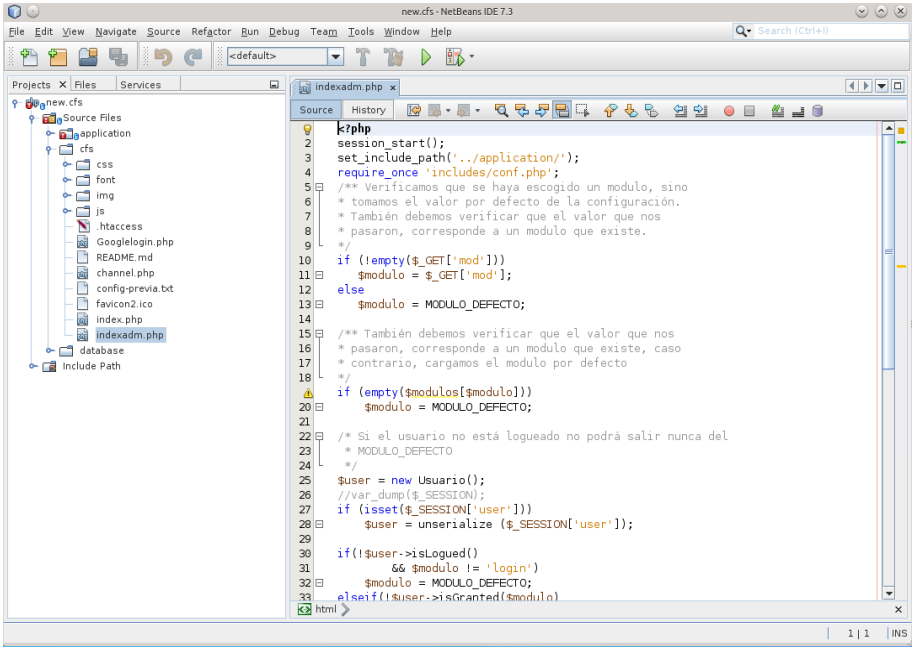

Figura 5.2 Captura de pantalla de NetBeans.

5.4. Git

Página oficial: http://git-scm.com/

Se trata de un software de control de versiones. Hoy en día es una herramienta básica para cualquier equipo de desarrollo. Da fiabilidad contra errores humanos y facilita la interoperabilidad de varios programadores contra un mismo proyecto.

El uso que se le ha dado en este proyecto es, básicamente, documental y preservación contra errores. Agregar funcionalidades nuevas mediante ramas[37] y así no tocar un código que ya funciona abastece de seguridad y fiabilidad a la aplicación. También facilita el trabajo en diferentes puestos (PC de sobremesa y portátil) utilizando un servidor remoto.

Git es un programa a nivel de consola y carece de interfaz gráfica propia. De todos modos, al ser código abierto, existen varias interfaces de usuario disponibles. Durante el proyecto solo se ha utilizado **gitg** para una representación gráfica, el resto de comandos se ejecutan desde consola. A continuación, en la imagen (Figura 5.3), vemos una captura de pantalla de dicho programa.

[37] La gran fuerza de este sistema es la creación de ramas (branch) que derivan de una versión ya funcional

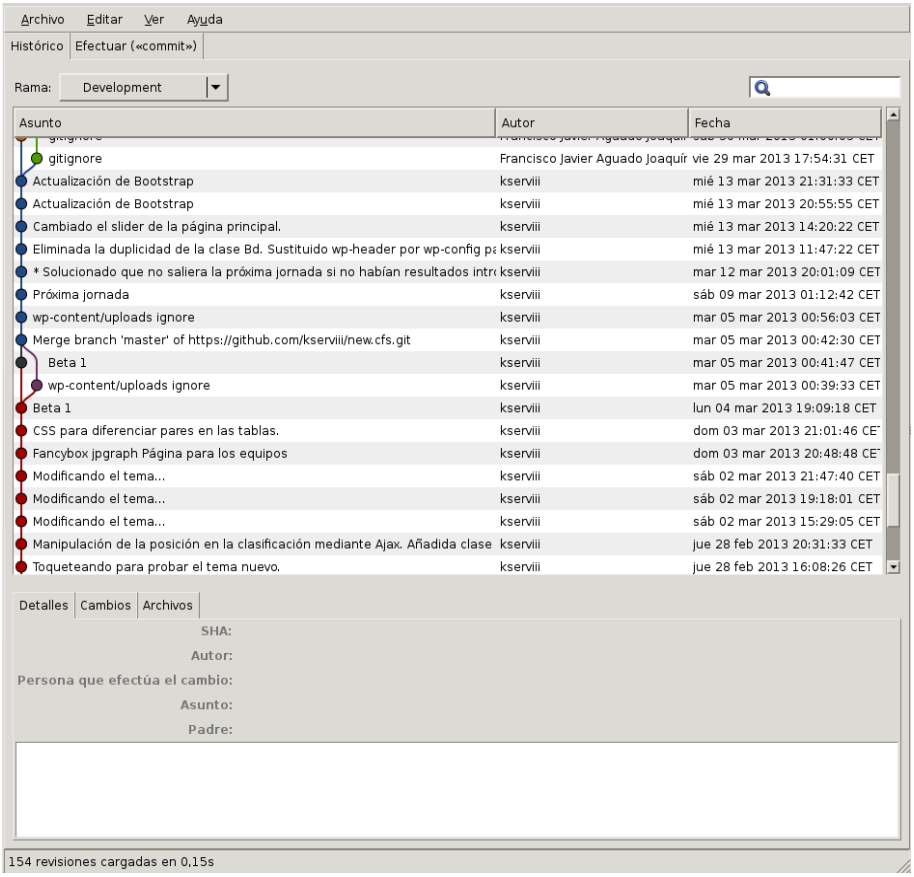

Figura 5.3 Captura del programa gitg.

5.4.1. GitHub

Página oficial: https://github.com/

Como se ha indicado, el uso que se le ha dado a Git, entre otros, es la posibilidad de trabajo entre varios puestos diferentes utilizando un servidor remoto.

GitHub es un servicio para repositorios en línea de Git. Es gratuito para repositorios de código abierto en el que cualquiera puede ver el código y clonar[38]. En este caso se ha optado, como ya se comentó anteriormente, por crear un repositorio privado (de pago) y así tener el control de quién puede o no ver el código.

A continuación se ve una captura de pantalla (Figura 5.4) del repositorio del proyecto.

[38] Acción de descargar a un PC un repositorio para poder modificar el código

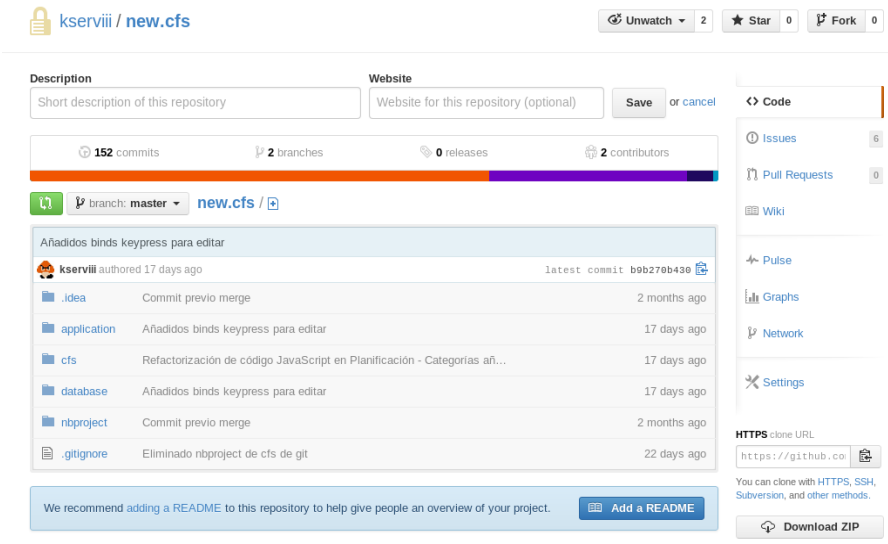

Figura 5.4 Captura de pantalla del servicio GitHub.

El despliegue en producción utilizando este tipo de servicios es un punto más a favor. En lugar de tener que volver a subir todo el proyecto solo se descargan los cambios realizados. Para eso es necesario que el servidor de producción tenga habilitado ssh[39] e instalado Git en el sistema.

[39]Conexión remota cifrada que se realiza sin interficie gráfica sino a través de terminal

6. Análisis y diseño

Durante el desarrollo de un proyecto es fácil perderse. Un buen análisis ayuda a tener una guía de la que no desviarse y no perder tiempo a cada paso para saber cómo continuar.

6.1. Base de Datos

6.1.1. Diseño entidades

En este apartado listamos todas las entidades que creamos en la base de datos representadas en forma de tabla. También se explican las relaciones que existen entre entidades y la función que realiza en el proyecto.

Entidad user

La primera entidad con la que nos encontramos, que es la entrada a nuestra aplicación, es la entidad **user** (ver Tabla 6.1) . Esta se encarga de almacenar los datos necesario para la autenticación dentro de nuestra aplicación.

user	
Campo	**Tipo**
id_{PK}	int(20) AI[40]
user	varchar(50)
cookie	varchar(255)
open_id	varchar(255)
facebook_id	bigint(255)
last_login	timestamp

Tabla 6.1 Entidad user.

Dado que la totalidad de los usuarios que van a utilizar la aplicación poseen cuenta de Google o Facebook; se opta por utilizar el sistema de autenticación de estas compañías para evitar registros innecesarios a los usuarios. Así, cuando el usuario se autentifica en la aplicación, se almacena el id de Google en el campo **open_id**, o bien el id de Facebook en el campo **facebook_id**; así como la dirección de correo electrónico en el campo **user**. Como se puede entrar por cualquier de los dos, y para facilitar posteriores referencias al usuario, se le da una id única dentro de nuestra base de datos. Una vez acceden, se guarda la fecha y la hora de acceso mediante un campo de tipo timestamp[41] llamado **last_login**. Para terminar, al autentificarse en el sistema se crea una **cookie**[42].

Entidad user_roles

Una vez un usuario se ha autentificado, es la tabla **user_roles** (ver Tabla 6.2) la que almacena qué rol ejercerá dentro de la aplicación. Dicho rol le otorgará unos privilegios u otros en función de cómo esté configurado.

[40] AutoIncrement o campo que se auto-incrementa cada vez que se inserta un nuevo elemento

[41] genera una fecha y hora automáticamente sin necesidad de programación alguna

[42] pequeña información enviada por un sitio web y almacenada en el navegador del usuario, de manera que el sitio web puede consultar la actividad previa del usuario

user_roles	
Campo	**Tipo**
user_id$_{PK}$	int(20) FK
role	int(2)

Tabla 6.2 Entidad user_roles.

Como se puede observar, se trata de una entidad muy sencilla. Lo único que hace es relacionar un usuario con los distintos roles que pueda ejercer. La lista de roles, dado que será difícil que cambien, se ha optado por almacenarla directamente en el código.

Código 6.1 Roles almacenados en la clase Roles.

```
<?php

class Roles {
    const ROLE_BASE =         9;
    const ROLE_ADMIN =        1;
    const ROLE_ENTRENADOR =   2;
    const ROLE_EDITOR =       3;
    const ROLE_RESULTADOS =   4;
    const ROLE_PLANIFICADOR = 5;
    ...
```

Entidad user_liga

Cuando a un usuario se le da el rol de *Entrenador*, este puede gestionar el equipo del que es entrenador. Esta entidad es la que almacena esa información. Relaciona al usuario con una liga a gestionar. La tabla la podemos ver a continuación (Tabla 6.3):

user_liga	
Campo	**Tipo**
id_user$_{PK}$	int(20) FK
id_liga	int(8)

Tabla 6.3 Entidad user_liga.

Entidad temporadas

Una vez tenemos el sistema de acceso a nuestra aplicación, lo primero que necesitamos para poder trabajar es una temporada en la que añadir datos.

Para tal fin se dispone de la entidad **temporadas** (ver Tabla 6.4).

temporadas	
Campo	**Tipo**
id_{PK}	int(4) AI
nombre	varchar(50)

Tabla 6.4 Entidad temporadas.

Se trata de una entidad que solo almacena el nombre de la temporada y así juntar el resto de datos bajo una misma.

Entidad categorias

Se tratan de los distintos niveles y edades en los que el club posee equipos. Por ejemplo: Senior A, Senior B, Juvenil A... La forma en que se almacenan los datos es similar a la entidad temporadas y lo podemos ver en la siguiente tabla (Tabla 6.5).

categorias	
Campo	**Tipo**
id_{PK}	int(8) AI
nombre	varchar(60)

Tabla 6.5 Entidad categorias.

Entidad ligas

Como cada temporada, los equipos participan en sus respectivas ligas. Antes de poder añadir los equipos que componen una cierta liga, se deben añadir todas las ligas en las que el club tendrá participación. Para almacenar esta información nos valemos de la entidad **ligas** (ver Tabla 6.6).

ligas	
Campo	**Tipo**
id_{PK}	int(8) AI
nombre	varchar(60)
grupo	varchar(2)
id_categoria	int(8) FK
id_temporada	int(4) FK
puntos_ganar	int(1)
puntos_empatar	int(1)
puntos_perder	int(1)
mostrar	tinyint(1)
orden	tinyint(2)

Tabla 6.6 Entidad ligas.

Normalmente todas las ligas tienen un **nombre** y un **grupo**. Las ligas pertenecen a una categoría (**id_categoria**) definida en la entidad **categoria** (Tabla 6.5). También pertenecen a una temporada (**id_temporada**) definida en la

entidad **temporada** (Tabla 6.4). En toda liga se otorgan unos puntos según el resultado de los partidos; estos puntos se almacenan en los campos: **puntos_ganar**, **puntos_empatar** y **puntos_perder**. Con el campo **mostrar** indicamos si queremos que se vea dicha liga en la parte pública de la aplicación o no; y con el campo **orden** indicamos el orden en que se muestran las ligas a la que les otorgamos visibilidad pública.

Entidad equipos

Una vez hemos añadido una temporada y agregado una liga en una cierta categoría, ya podemos añadirle equipos a dicha liga para poder configurar la competición. La información de los equipos se almacena en la entidad **equipos** (Tabla 6.7).

equipos	
Campo	**Tipo**
id_{PK}	int(50) AI
nombre	varchar(60)
diminutivo	varchar(60)
dia	int(1)
hora	time
id_liga	int(8) FK
id_temporada	int(4) FK
nuestro	tinyint(1)
foto	varchar(70)

Tabla 6.7 Entidad equipos.

Como es lógico, todo equipo posee un **nombre**. El campo **diminutivo** sirve para expresar un nombre más corto para zonas de la parte pública en el que no se pudiera mostrar el nombre completo por falta de espacio. Todo equipo también pertenece a la liga que disputa, siendo representada por el campo **id_liga** que apunta a la entidad ligas (Tabla 6.6); también pertenece a la temporada en curso, siendo el campo **id_temporada** el encargado de almacenar un id de la entidad temporada (Tabla 6.4). El campo **dia** nos indica si el equipo juega en sábado (valor 0) o domingo (valor 1) y el campo **hora** su hora de partido habitual. Para saber qué equipo pertenece al club de entre todos los que participan en una liga, nos valemos de **nuestro**. Para terminar, si el equipo tiene una foto ligada, en el campo **foto** tendremos el nombre del archivo almacenado en el disco duro.

Entidad jornadas

El desarrollo de una liga está dividido en jornadas. En cada jornada se disputan una serie de enfrentamientos. Para almacenar la forma en que los equipos juegan los partidos se utiliza la entidad **jornadas** (Tabla 6.8).

jornadas	
Campo	**Tipo**
id_{PK}	int(12) AI
id_liga	int(8) FK
id_visitante	int(50) FK
id_local	int(50) FK
descansa	tinyint(1)
fecha	date

Tabla 6.8 Entidad jornadas.

En esta entidad relacionamos un equipo local **id_local** con un equipo visitante **id_visitante** (Tabla 6.7) pertenecientes a una liga **id_liga** (Tabla 6.6) que se enfrentarán el día marcado por el campo **fecha**. En caso que al equipo le toque descansar esa jornada, se indicará utilizando el campo **descansa**.

Entidad resultados

Una vez disputado el enfrentamiento dictado por la entidad jornadas (Tabla 6.8), los resultados se almacenan en esta entidad (Tabla 6.9).

resultados	
Campo	**Tipo**
$id\_jornada_{PK}$	int(12) AI
rlocal	int(2)
rvisitante	int(2)
motivo	text
aplazado	tinyint(1)
fecha_aplazado	date
pendiente	tinyint(1)

Tabla 6.9 Entidad resultados.

Los resultados hacen referencia a las jornadas mediante **id_jornada** (Tabla 6.8). En esta entidad, como su nombre indica, almacenamos el resultado del enfrentamiento; en **rlocal** guardamos el resultado del equipo que juega como local (o en casa) y en **rvisitante** el visitante. Puede darse que el partido sea aplazado. Así, es el campo **aplazado** el que nos lo indica; también se almacena el **motivo** del aplazamiento y la **fecha_aplazado** en la que se disputará el partido. Por último, el campo **pendiente** nos indica que los resultados de dicha jornada ya han sido introducidos pero ese, en particular, está pendiente de introducir.

Entidad clasificaciones

Una vez introducidos los resultados toca calcular los puntos, goles y partidos que ha ganado, empatado y perdido cada equipo. Estos datos se almacenan en esta entidad (Tabla 6.10).

clasificaciones	
Campo	**Tipo**
id_{PK}	int(100) AI
id_liga	int(8) FK
id_equipo	int(50) FK
num_jornada	int(2)
posicion	int(2)
gf	int(3)
gc	int(3)
pg	int(3)
pe	int(3)
pp	int(3)
puntos	int(3)

Tabla 6.10 Entidad clasificaciones.

Para cada entrada en la entidad clasificaciones se guarda el equipo al que pertenece en **id_equipo** (Tabla 6.7) y la liga en **id_liga** (Tabla 6.6). El campo **num_jornada** nos hace referencia a la jornada en la que se dan esos datos, así se pueden ver la evolución durante la temporada. El campo **posicion** nos indica en qué posición, de entre el resto de equipos de la liga, se encuentra el equipo. En el campo **puntos** tenemos los puntos acumulados en la jornada por un cierto equipo. El resto de datos que se guardan para cada jornada y equipo son:

1. **gf**: goles a favor
2. **gc**: goles en contra
3. **pg**: partidos ganados
4. **pe**: partidos empatados
5. **pp**: partidos perdidos

Entidad noticias

Las noticias que se redactan para información de socios y seguidores se almacenan aquí. En la siguiente tabla se puede ver la estructura de esta entidad (Tabla 6.11).

noticias	
Campo	**Tipo**
id_{PK}	int(20) AI
title	varchar(255)
post	text
slug	varchar(255)
author_id	int(20)
fecha	datetime
id_liga	int(8) FK

Tabla 6.11 Entidad noticias.

Una noticia ha de ser creada por un usuario, ese dato se guarda en el campo **author_id** y hace referencia a los usuarios (Tabla 6.1). La noticia puede ser con temática referente a un equipo en concreto; en cuyo caso se hará referencia a la liga (Tabla 6.6) en la que participan utilizando el campo **id_liga**. El resto de datos son habituales en los blogs. El primero es el título de la noticia almacenado en **title**; del título se genera un slug[43] que se almacena en **slug**. El contenido de la noticia es almacenado en el campo **post**. Por último, se guarda en el campo **fecha** la fecha en la que se crea la noticia.

Entidad config

Se trata de una entidad auxiliar donde podemos almacenar variables para recuperarlas en sesiones posteriores. La estructura es tan básica como se muestra en la siguiente tabla (Tabla 6.12).

config	
Campo	**Tipo**
$campo_{PK}$	varchar(255)
id_campo	int(100)
valor_campo	varchar(255)

Tabla 6.12 Entidad config.

En **campo** tenemos un nombre por el que podemos reconocer la variable que hemos almacenado en la base de datos. El campo **id_campo** es para almacenar un id en caso que hagamos referencia a algún valor de otra entidad. Por último, el campo **valor_campo** es donde queda almacenado el valor que se le quiera dar a la variable. Por ejemplo (Tabla 6.13):

[43] se trata del título quitando espacios y caracteres raros para poder usarlo de url y así ser una ayuda para el posicionamiento web

Ejemplo	
Campo	**Valor**
campo	temporada
id_campo	12
valor_campo	Temporada 2012 - 2013

Tabla 6.13 Ejemplo de la entidad config.

Entidad notificaciones

Se trata de una entidad donde se almacenan las notificaciones que los administradores puedan mandar al resto de usuarios. La estructura la podemos ver en la tabla (Tabla 6.14).

config	
Campo	**Tipo**
id_{PK}	int(10)
role	tinyint(2)
mensaje	text
tipo	varchar(10)
timestamp	varchar(255)

Tabla 6.14 Entidad config.

En **role** tenemos el rol al que va destinada la notificación. En **mensaje** es obvio que tenemos el mensaje a notificar. En **tipo** se almacena el tipo de mensaje: satisfactorio, error, alerta... **timestamp** es una marca de tiempo que genera la base de datos automáticamente.

Entidad log

En esta entidad es donde se almacena el comportamiento que ejerce el usuario sobre la aplicación. Aquí podemos ver quién entra al sistema, a qué hora, desde qué IP y qué hace. La estructura de la entidad la podemos ver en la tabla (Tabla 6.15).

log	
Campo	**Tipo**
id_{PK}	int(100)
id_user	int(20) FK
ip	int(10)
browser	varchar(255)
fecha	datetime
accion	varchar(255)

Tabla 6.15 Entidad log.

Lo primero que se almacena en esta entidad es quién hace algo, esto se almacena en el campo **id_user** que hace referencia al usuario (Tabla 6.1). La información que se almacena del usuario es:

1. **ip**: la IP desde la que accede y realiza la acción.
2. **browser**: el user agent[44] que utiliza.
3. **fecha**: la fecha y hora cuándo se realiza la acción.
4. **accion**: la acción que se lleva a cabo.

6.1.2. Diagrama relacional

En la siguiente figura (Figura 6.1) podemos ver la relaciones entre las entidades de la base de datos.

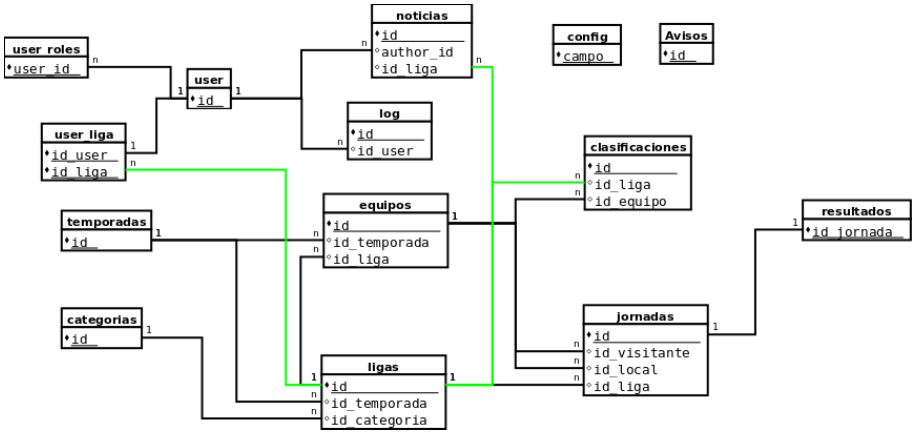

Figura 6.1 Diagrama relacional de la base de datos.

En el diagrama solo se muestran las claves primarias y foráneas dado que anteriormente ya se han detallado los campos de todas las entidades de la base de datos.

6.2. Zona de administración (backend)

El diseño de la zona de administración, o backend, se realiza utilizando diagramas de casos de uso. Estos diagramas indican, dependiendo del usuario que trate, las posibles acciones que este puede realizar contra el sistema.

6.2.1. Usuario no autentificado

Un usuario no autentificado es un usuario que no sabemos quién es. El primer paso antes de poder utilizar una aplicación, es decirle a la misma quién eres. Esta acción se describe en el siguiente diagrama (Figura 6.2):

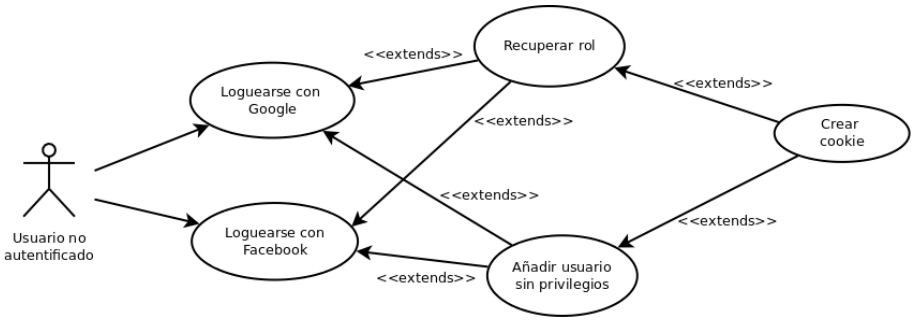

Figura 6.2 Diagrama de usuario no autentificado.

Como se puede apreciar, un usuario no autentificado solo dispone de dos acciones posibles:

Loguearse con Google

Accede al sistema usando su cuenta de Google.

Flujo de eventos:

1. El usuario selecciona la opción de Google.
2. El sistema le redirecciona al sistema de autenticación de Google para que introduzca sus credenciales.
3. El sistema recoge las credenciales.
4. Si el usuario no se ha logueado nunca en el sistema, añade el usuario a la base de datos guardando su dirección de correo y el ID de Google.
5. Si el usuario ha marcado la opción de *Recordarme*, se genera una cookie en el navegador del usuario.
6. El sistema carga la página principal.

Flujos alternativos:

En el paso 4, si el usuario ya se ha logueado antes:

4. se recupera el rol que le ha asignado un administrador.

En el paso 5, si el usuario no ha marcado la opción *Recordarme*:

5. no se genera cookie y se elimina cualquier posible resto de alguna anterior.

Loguearse con Facebook

Accede al sistema usando su cuenta de Facebook.

Flujo de eventos:

1. El usuario selecciona la opción de Facebook.

2. El sistema le redirecciona al sistema de autenticación de Facebook para que introduzca sus credenciales.
3. El sistema recoge las credenciales.
4. Si el usuario no se ha logueado nunca en el sistema, añade el usuario a la base de datos guardando su dirección de correo y el ID de Facebook.
5. Si el usuario ha marcado la opción de *Recordarme*, se genera una cookie en el navegador del usuario.
6. El sistema carga la página principal.

Flujos alternativos:

En el paso 4, si el usuario ya se ha logueado antes:

4. se recupera el rol que le ha asignado un administrador.

En el paso 5, si el usuario no ha marcado la opción *Recordarme*:

5. no se genera cookie y se elimina cualquier posible resto de alguna anterior.

6.2.2. Usuario con rol editor

Un usuario con rol editor dispondrá de las acciones que se detallan en el siguiente diagrama (Figura 6.3):

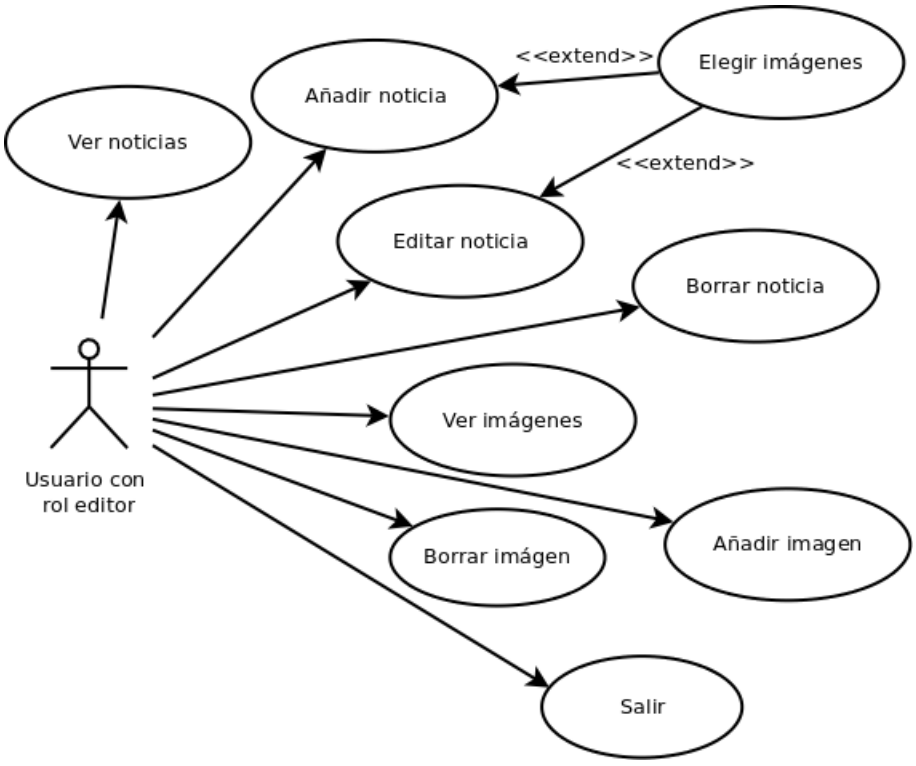

Figura 6.3 Diagrama rol editor.

Ver noticias

Se accede a un listado de las noticias almacenadas en la aplicación.

Flujo de eventos:

1. El usuario accede a la opción de *noticias*.
2. El sistema muestra una lista de noticias con botones de acciones.

Añadir noticia

Escribir una noticia para que aparezca en la parte pública.

Flujo de eventos:

1. El usuario accede a la opción de *noticias*.
2. El sistema muestra una lista de noticias con botones de acciones.
3. El usuario accede a añadir noticia.
4. El sistema muestra un editor wysiwyg.
5. El usuario escribe la noticia.
6. El usuario le da al botón guardar.
7. El sistema guarda la noticia.
8. El sistema vuelve a la opción de *noticias* con la lista actualizada.

Editar noticia

Editar una noticia ya existente.

Flujo de eventos:

1. El usuario accede a la opción de *noticias*.
2. El sistema muestra una lista de noticias con botones de acciones.
3. El usuario accede a la opción *Editar* de una noticia de la lista.
4. El sistema muestra un editor wysiwyg con la noticia cargada.
5. El usuario edita la noticia.
6. El usuario le da al botón guardar.
7. El sistema guarda la noticia.
8. El sistema vuelve a la opción de *noticias* con la lista actualizada.

Elegir imagen

Se elige una imagen de las que hay subidas al servidor:

Flujo de eventos:

1. El usuario accede a la opción *imagen* del editor wysiwyg.
2. El sistema muestra una ventana de propiedades de imagen.
3. El usuario accede a la opción *Ver servidor*.
4. El sistema muestra las imágenes del servidor.
5. El usuario selecciona una imagen.
6. El sistema recupera la ruta de la imagen y la añade a la ventana de propiedades de imagen.
7. El usuario le da a *Aceptar* antes pudiendo editar las propiedades de la imagen.
8. El sistema la inserta en el editor wysiwyg.
9. El usuario puede seguir añadiendo o editando la noticia.

Borrar noticia

Borrar una noticia almacenada:

Flujo de eventos:

1. El usuario accede a la opción de *noticias*.
2. El sistema muestra una lista de noticias con botones de acciones.
3. El usuario elige la opción *Borrar* de una noticia de la lista.
4. El sistema muestra una confirmación para borrar la noticia.
5. El usuario acepta la confirmación.
6. El sistema borra la noticia de la base de datos.
7. El sistema vuelve a la opción de *noticias* con la lista actualizada.

Flujos alternativos:

En el paso 6, si el usuario no acepta la confirmación del paso 5:

6. El sistema vuelve a la opción de *noticias* sin modificaciones.

Ver imágenes

Galería con todas las imágenes subidas al servidor:

Flujo de eventos:

1. El usuario accede a la opción de *imágenes*.
2. El sistema muestra una galería de imágenes con botones de acciones.

Añadir imagen

El usuario carga una imagen en el servidor:

Flujo de eventos:

1. El usuario accede a la opción de *imágenes*.
2. El sistema muestra una galería de imágenes con botones de acciones.
3. El usuario elige la opción *Elegir archivos*.
4. El sistema muestra un navegador para elegir la imagen que se quiera cargar.
5. El usuario elige la imagen deseada.
6. El sistema la guarda en varios formatos.
7. El sistema actualiza la vista de imágenes.

Flujos alternativos:

En el paso 6, si el usuario cancela en lugar de elegir imagen en el paso 5:

6. El sistema muestra la galería sin modificar.

Borrar imagen

Borrar una imagen del servidor:

Flujo de eventos:

1. El usuario accede a la opción de *imágenes*.
2. El sistema muestra una galería de imágenes con botones de acciones.
3. El usuario elige la opción *Borrar* de la imagen que desea borrar.
4. El sistema una confirmación.
5. El usuario acepta la confirmación.
6. El sistema borra la imagen.
7. El sistema actualiza la vista de imágenes.

Flujos alternativos:

En el paso 6, si el usuario cancela en el paso 5:

6. El sistema muestra la galería sin modificar.

Salir

Salir de la aplicación:

Flujo de eventos:

1. El usuario elige la opción *salir*.
2. El sistema elimina las variables de sesión.
3. El sistema elimina la cookie.
4. El sistema sale de la aplicación.

6.2.3. Usuario con rol entrenador

Un usuario con rol entrenador dispondrá de las acciones que se detallan en el siguiente diagrama (Figura 6.4):

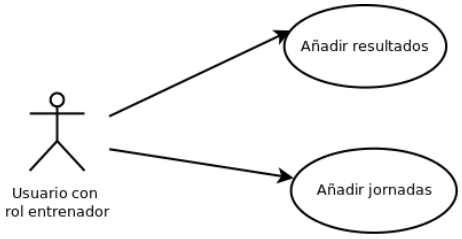

Figura 6.4 Diagrama rol entrenador

Añadir resultados

Añadir resultados de partidos ya disputados.

Flujo de eventos:

1. El usuario accede a la opción *resultados*.
2. El sistema muestra las ligas a las que tiene acceso.
3. El usuario elige la liga en la que quiere añadir resultados.
4. El sistema le muestra la última jornada a la que le faltan los resultados.
5. El usuario inserta los resultados.
6. El usuario le da al botón guardar.
7. El sistema guarda los resultados.
8. El sistema calcula las clasificaciones.
9. El sistema pasa a la siguiente jornada.

Flujos alternativos:

En el paso 5, si el usuario no quisiera esa jornada:

5. El usuario elige la jornada que desea.
6. El usuario inserta los resultados.
7. El usuario le da al botón guardar.
8. El sistema guarda los resultados.
9. El sistema calcula las clasificaciones.
10. El sistema pasa a la siguiente jornada.

Añadir jornadas

Añadir jornadas para la confección del calendario deportivo.

Flujo de eventos:

1. El usuario accede a la opción *jornadas*.
2. El sistema muestra las ligas a las que tiene acceso.
3. El usuario elige la liga en la que quiere añadir jornadas.
4. El sistema le muestra la última jornada a la que le faltan los enfrentamientos.
5. El usuario inserta los enfrentamientos.
6. El usuario pone las fechas del partido de ida y vuelta.
7. El usuario le da al botón guardar.
8. El sistema guarda los resultados.
9. El sistema pasa a la siguiente jornada.

Salir

Ver en sección 6.2.2

6.2.4. Usuario con rol resultados

Un usuario con rol entrenador dispondrá de las acciones que se detallan en el siguiente diagrama (Figura 6.5):

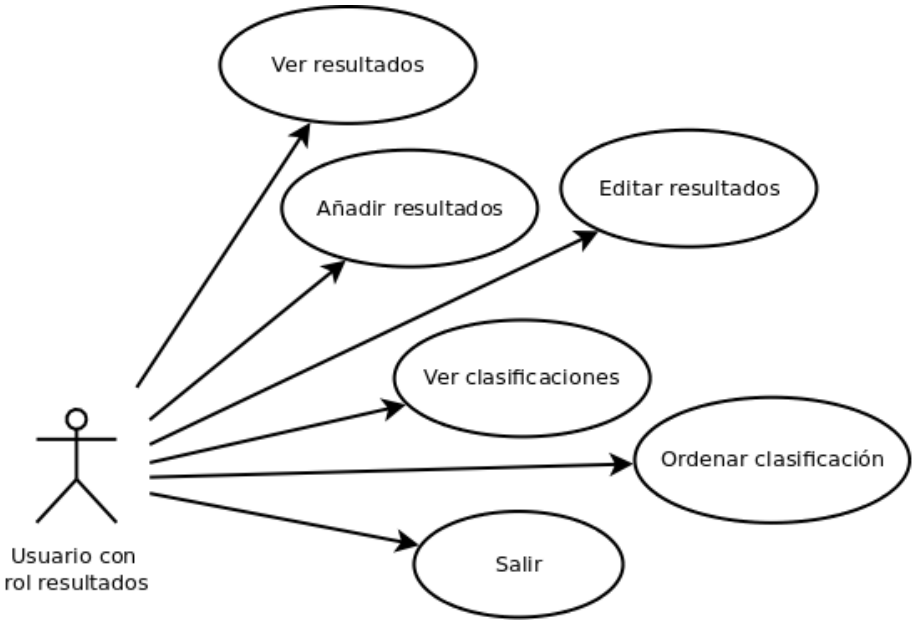

Figura 6.5 Diagrama rol resultados.

Ver resultados

Ver resultados de partidos anteriores.

Flujo de eventos:

1. El usuario accede a la opción *resultados*.
2. El sistema muestra las ligas de la temporada de trabajo.
3. El usuario elige la liga en la que quiere añadir resultados.
4. El sistema le muestra la última jornada a la que le faltan los resultados.
5. El usuario elige la jornada que desea ver.
6. El sistema muestra los resultados de la jornada seleccionada.

Flujos alternativos:

En el paso 6, si la jornada no ha sido introducida:

6. El sistema ofrece la posibilidad de añadir la jornada.

En el paso 6, si la jornada no tiene resultados:

6. El sistema ofrece la posibilidad de añadir los resultados.

Añadir resultados

Añadir resultados de partidos ya disputados.

Flujo de eventos:

1. El usuario accede a la opción *resultados*.
2. El sistema muestra las ligas de la temporada de trabajo.
3. El usuario elige la liga en la que quiere añadir resultados.
4. El sistema le muestra la última jornada a la que le faltan los resultados.
5. El usuario inserta los resultados.
6. El usuario le da al botón guardar.
7. El sistema guarda los resultados.
8. El sistema calcula las clasificaciones.
9. El sistema pasa a la siguiente jornada.

Flujos alternativos:

En el paso 5, si el usuario no quisiera esa jornada:

5. El usuario elige la jornada que desea.
6. El usuario inserta los resultados.
7. El usuario le da al botón guardar.
8. El sistema guarda los resultados.
9. El sistema calcula las clasificaciones.
10. El sistema pasa a la siguiente jornada.

Editar resultados

Editar resultados de partidos ya disputados.

Flujo de eventos:

1. El usuario accede a la opción *resultados*.
2. El sistema muestra las ligas de la temporada de trabajo.
3. El usuario elige la liga en la que quiere añadir resultados.
4. El sistema le muestra la última jornada a la que le faltan los resultados.
5. El usuario elige la jornada que desea editar.
6. El usuario cambia los resultados.
7. El usuario le da al botón Actualizar.
8. El sistema guarda los resultados.
9. El sistema recalcula las clasificaciones de la jornada y posteriores.
10. El sistema muestra la jornada ya editada.

Ver clasificaciones

Ver las clasificaciones que se han calculado.

Flujo de eventos:

1. El usuario accede a la opción *clasificaciones*.
2. El sistema muestra las ligas de la temporada de trabajo.
3. El usuario elige la liga de la que quiere ver las clasificaciones.
4. El sistema le muestra la última clasificación de los resultados añadidos.
5. El usuario elige la jornada que desea ver.
6. El sistema le muestra la clasificación elegida.

Flujos alternativos:

En el paso 5, si los resultados no han sido introducidos:

6. El sistema ofrece la posibilidad de añadir resultados.

Ordenar clasificaciones

Ordena la posición del equipo dentro de la clasificación.

Flujo de eventos:

1. El usuario accede a la opción *clasificaciones*.
2. El sistema muestra las ligas de la temporada de trabajo.
3. El usuario elige la liga de la que quiere ver las clasificaciones.
4. El sistema le muestra la última clasificación de los resultados añadidos.
5. El usuario elige la jornada que desea ver.
6. El sistema le muestra la clasificación elegida.
7. El usuario arrastra el equipo hasta la posición deseada.
8. El sistema cambia la posición del equipo.

Flujos alternativos:

En el paso 5, si los resultados no han sido introducidos:

6. El sistema ofrece la posibilidad de añadir resultados.

Salir

Ver en sección 6.2.2

6.2.5. Usuario con rol planificador

Un usuario con rol planificador dispondrá de las acciones que se detallan en el siguiente diagrama (Figura 6.6):

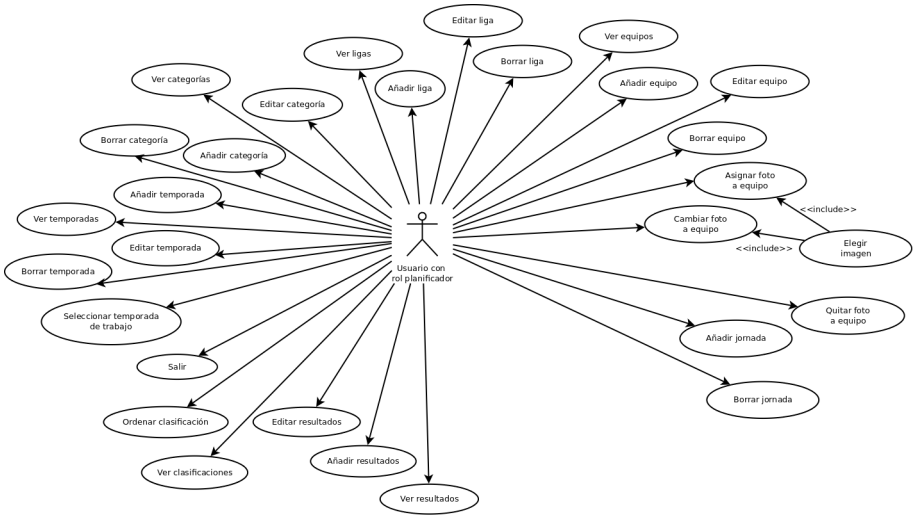

Figura 6.6 Diagrama rol planificador.

Ver temporadas

Ver un listado de las temporadas almacenadas.

Flujo de eventos:

1. El usuario accede a la opción *temporadas*.
2. El sistema muestra la lista de las temporadas con botones de acción.

Seleccionar temporada de trabajo

Seleccionar la temporada en la que se realizarán las inserciones o cambios.

Flujo de eventos:

1. El usuario accede a la opción *temporadas*.
2. El sistema muestra la lista de las temporadas con botones de acción.
3. El usuario selecciona la temporada con la que desea trabajar.
4. El sistema asigna esa temporada para las inserciones o cambios.

Añadir temporada

Añadir una temporada nueva.

Flujo de eventos:

1. El usuario accede a la opción *temporadas*.
2. El sistema muestra la lista de las temporadas con botones de acción.
3. El usuario le asigna un nombre.
4. El usuario elije la opción *añadir*.
5. El sistema añade la temporada.
6. El sistema muestra la lista actualizada.

Editar temporada

Editar el nombre de una temporada.

Flujo de eventos:

1. El usuario accede a la opción *temporadas*.
2. El sistema muestra la lista de las temporadas con botones de acción.
3. El usuario elije la opción *Editar*.
4. El sistema cambia el nombre por un cuadro de texto editable.
5. El usuario edita el nombre.
6. El usuario elije la opción *Guardar*.
7. El sistema muestra la lista actualizada.

Flujos alternativos:

En el paso 6:

6. El usuario elige la opción *Cancelar*.
7. El sistema muestra la lista anterior.

Borrar temporada

Borrar una temporada de la lista.

Flujo de eventos:

1. El usuario accede a la opción *temporadas*.
2. El sistema muestra la lista de las temporadas con botones de acción.
3. El usuario elije la opción *Borrar*.
4. El sistema lanza una confirmación.
5. El usuario confirma.
6. El sistema comprueba que no sea la temporada actual de trabajo.

7. El sistema comprueba si hay ligas asignadas a la temporada.
8. El sistema borra la temporada.
9. El sistema muestra la lista actualizada.

Flujos alternativos:

En el paso 7, en caso que si sea la temporada actual de trabajo:

7. El sistema lanza un mensaje de error.
8. El sistema muestra la lista anterior.

En el paso 8, en caso que si hayan ligas asignadas:

8. El sistema lanza un mensaje de error.
9. El sistema muestra la lista anterior.

Ver categorías

Ver un listado de las categorías almacenadas.

Flujo de eventos:

1. El usuario accede a la opción *categorías*.
2. El sistema muestra la lista de las categorías con botones de acción.

Añadir categoría

Añadir una categoría nueva.

Flujo de eventos:

1. El usuario accede a la opción *categorías*.
2. El sistema muestra la lista de las categorías con botones de acción.
3. El usuario le asigna un nombre.
4. El usuario elije la opción *añadir*.
5. El sistema añade la categoría.
6. El sistema muestra la lista actualizada.

Editar categoría

Editar el nombre de una categoría.

Flujo de eventos:

1. El usuario accede a la opción *categorías*.
2. El sistema muestra la lista de las categorías con botones de acción.
3. El usuario elije la opción *Editar*.
4. El sistema cambia el nombre por un cuadro de texto editable.
5. El usuario edita el nombre.
6. El usuario elije la opción *Guardar*.
7. El sistema muestra la lista actualizada.

Flujos alternativos:

En el paso 6:

6. El usuario elige la opción *Cancelar*.
7. El sistema muestra la lista anterior.

Borrar categoría

Borrar una categoría de la lista.

Flujo de eventos:

1. El usuario accede a la opción *categorías*.
2. El sistema muestra la lista de las categorías con botones de acción.
3. El usuario elije la opción *Borrar*.
4. El sistema lanza una confirmación.
5. El usuario confirma.
6. El sistema comprueba si hay ligas asignadas a la categoría.
7. El sistema borra la categoría.
8. El sistema muestra la lista actualizada.

Flujos alternativos:

En el paso 7, en caso que si hayan ligas asignadas:

7. El sistema lanza un mensaje de error.
8. El sistema muestra la lista anterior.

Ver ligas

Ver un listado de las ligas de la temporada de trabajo.

Flujo de eventos:

1. El usuario accede a la opción *ligas*.
2. El sistema muestra la lista de las ligas con botones de acción.

Añadir liga

Añadir una liga nueva.

Flujo de eventos:

1. El usuario accede a la opción *ligas*.
2. El sistema muestra la lista de las categorías con botones de acción.
3. El usuario elige la opción *Añadir*.
4. El sistema muestra una ventana para que introduzca los datos.
5. El usuario introduce los datos.
6. El usuario elige la opción *Añadir*.
7. El sistema añade la liga.
8. El sistema muestra la lista actualizada.

Flujos alternativos:

En el paso 6:

6. El usuario elige la opción *Cancelar*.
7. El sistema muestra la lista.

En el paso 7, si el usuario introduce algún dato incorrecto:

7. El sistema muestra un error.
8. El sistema espera a que el error se corrija.

Editar liga

Editar una liga existente.

Flujo de eventos:

1. El usuario accede a la opción *ligas*.
2. El sistema muestra la lista de las categorías con botones de acción.
3. El usuario elige la opción *Editar*.

4. El sistema muestra una ventana con los datos de la liga.
5. El usuario edita los datos.
6. El usuario elige la opción *Editar*.
7. El sistema guarda los cambios.
8. El sistema muestra la lista actualizada.

Flujos alternativos:

En el paso 6:

6. El usuario elige la opción *Cancelar*.
7. El sistema muestra la lista.

En el paso 7, si el usuario introduce algún dato incorrecto:

7. El sistema muestra un error.
8. El sistema espera a que el error se corrija.

Borrar liga

Borrar una liga de la lista.

Flujo de eventos:

1. El usuario accede a la opción *ligas*.
2. El sistema muestra la lista de las ligas con botones de acción.
3. El usuario elije la opción *Borrar*.
4. El sistema lanza una confirmación.
5. El usuario confirma.
6. El sistema comprueba si hay equipos asignados a la liga.
7. El sistema borra la liga.
8. El sistema muestra la lista actualizada.

Flujos alternativos:

En el paso 7, en caso que si hayan equipos asignados:

7. El sistema lanza un mensaje de error.
8. El sistema muestra la lista anterior.

Ver equipos

Ver un listado de las equipos de una liga.

Flujo de eventos:

1. El usuario accede a la opción *equipos*.
2. El sistema muestra las ligas de la temporada de trabajo.
3. El usuario elige la liga de la que desea ver los equipos.
4. El sistema muestra la lista de los equipos con botones de acción.

Añadir equipo

Añadir un equipo nuevo a una liga.

Flujo de eventos:

1. El usuario accede a la opción *equipos*.
2. El sistema muestra las ligas de la temporada de trabajo.
3. El usuario elige la liga de la que desea ver los equipos.
4. El sistema muestra la lista de los equipos con botones de acción.
5. El usuario elige la opción *Añadir*.
6. El sistema comprueba si la liga ha sido iniciada.
7. El sistema muestra una ventana para que introduzca los datos.
8. El usuario introduce los datos.
9. El usuario elige la opción *Añadir*.
10. El sistema añade la liga.
11. El sistema muestra la lista actualizada.

Flujos alternativos:

En el paso 6, si el sistema detecta jornadas introducidas:

7. El sistema muestra un mensaje de error y cancela la acción.

En el paso 9, si el usuario introduce algún dato incorrecto:

9. El sistema muestra un error.
10. El sistema espera a que el error se corrija.

En el paso 9:

9. El usuario elige la opción *Cancelar*.
10. El sistema muestra la lista.

Editar equipo

Editar un equipo de una liga.

Flujo de eventos:

1. El usuario accede a la opción *equipos*.
2. El sistema muestra las ligas de la temporada de trabajo.
3. El usuario elige la liga de la que desea ver los equipos.
4. El sistema muestra la lista de los equipos con botones de acción.
5. El usuario elige la opción *Editar*.
6. El sistema muestra una ventana con los datos del equipo.
7. El usuario edita los datos.
8. El usuario elige la opción *Editar*.
9. El sistema guarda los cambios.
10. El sistema muestra la lista actualizada.

Flujos alternativos:

En el paso 8:

6. El usuario elige la opción *Cancelar*.
7. El sistema muestra la lista.

En el paso 9, si el usuario introduce algún dato incorrecto:

9. El sistema muestra un error.
10. El sistema espera a que el error se corrija.

Borrar equipo

Borrar un equipo de una liga.

Flujo de eventos:

1. El usuario accede a la opción *equipos*.
2. El sistema muestra las ligas de la temporada de trabajo.
3. El usuario elige la liga de la que desea ver los equipos.
4. El sistema muestra la lista de los equipos con botones de acción.
5. El usuario elije la opción *Borrar*.
6. El sistema lanza una confirmación.

7. El usuario confirma.
8. El sistema comprueba si la liga ha sido iniciada.
9. El sistema borra el equipo.
10. El sistema muestra la lista actualizada.

Flujos alternativos:

En el paso 9, en el caso que la liga haya sido iniciada:

7. El sistema lanza un mensaje de error.
8. El sistema muestra la lista anterior.

Asignar foto a equipo

Asignar una foto a un equipo.

Flujo de eventos:

1. El usuario accede a la opción *equipos*.
2. El sistema muestra las ligas de la temporada de trabajo.
3. El usuario elige la liga de la que desea ver los equipos.
4. El sistema muestra la lista de los equipos con botones de acción.
5. El usuario elije la opción *Seleccionar*.
6. El sistema lanza una ventana con la galería de imágenes.
7. El usuario elige imagen.
8. El sistema asigna la imagen al equipo.

Flujos alternativos:

En el paso 7:

7. El usuario cierra la ventana sin elegir.
8. El sistema muestra la lista anterior.

Cambiar foto a equipo

Cambiar una foto a un equipo.

Flujo de eventos:

1. El usuario accede a la opción *equipos*.
2. El sistema muestra las ligas de la temporada de trabajo.
3. El usuario elige la liga de la que desea ver los equipos.

4. El sistema muestra la lista de los equipos con botones de acción.
5. El usuario elije la opción *Cambiar*.
6. El sistema lanza una ventana con la galería de imágenes.
7. El usuario elige imagen.
8. El sistema asigna la imagen al equipo.

Flujos alternativos:

En el paso 7:

7. El usuario cierra la ventana sin elegir.
8. El sistema muestra la lista anterior.

Quitar foto a equipo

Quitar una foto a un equipo.

Flujo de eventos:

1. El usuario accede a la opción *equipos*.
2. El sistema muestra las ligas de la temporada de trabajo.
3. El usuario elige la liga de la que desea ver los equipos.
4. El sistema muestra la lista de los equipos con botones de acción.
5. El usuario elije la opción *Quitar foto*.
6. El sistema lanza una confirmación.
7. El usuario acepta.
8. El sistema desasigna la imagen al equipo.

Flujos alternativos:

En el paso 6:

7. El usuario cancela la confirmación.
8. El sistema muestra la lista anterior.

Añadir jornada

Añadir una jornada nueva a una liga.

Flujo de eventos:

1. El usuario accede a la opción *Añadir jornada*.
2. El sistema muestra las ligas de la temporada de trabajo.
3. El usuario elige la liga de la que desea añadir la jornada.
4. El sistema muestra la lista de equipos para los emparejamientos.
5. El usuario elige todas las parejas de los partidos de la jornada.
6. El usuario elige la fecha de la ida y de la vuelta.
7. El usuario elige la opción *Guardar jornada*.
8. El sistema comprueba que las fechas sean correctas.
9. El sistema añade la jornada.
10. El sistema la siguiente jornada a introducir.

Flujos alternativos:

En el paso 8, si las fechas no son correctas:

9. El sistema muestra un mensaje de error y espera la corrección.

Borrar jornada

Borrar una jornada y todas las posteriores de una liga.

Flujo de eventos:

1. El usuario accede a la opción *Borrar jornada*.
2. El sistema muestra las ligas de la temporada de trabajo.
3. El usuario elige la liga de la que desea borrar la jornada.
4. El sistema muestra la lista de las jornadas introducidas.
5. El usuario elige la jornada a borrar.
6. El sistema lanza confirmación.
7. El usuario acepta la confirmación.
8. El sistema borra la jornada y todas las posteriores.

Flujos alternativos:

En el paso 7:

7. El usuario cancela la confirmación.
8. El sistema muestra la lista de las jornadas introducidas.

Ver resultados

Ver en sección 6.2.4

Añadir resultados

Ver en sección 6.2.4

Editar resultados

Ver en sección 6.2.4

Ver clasificaciones

Ver en sección 6.2.4

Ordenar clasificaciones

Ver en sección 6.2.4

Salir

Ver en sección 6.2.2

6.2.6. Usuario con rol administrador

Un usuario con rol administrador dispondrá de las acciones que se detallan en el siguiente diagrama (Figura 6.7):

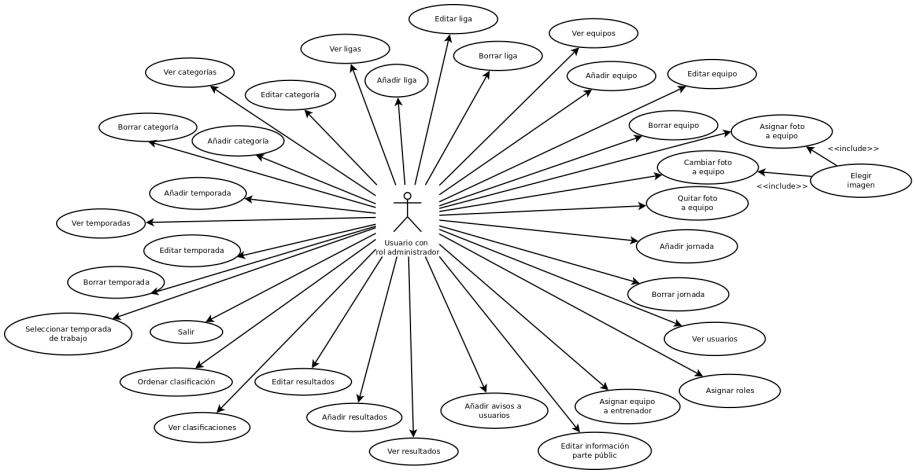

Figura 6.7 Diagrama rol administrador.

Ver usuarios

Ver un listado de las usuarios almacenados en la base de datos.

Flujo de eventos:

1. El usuario accede a la opción *usuarios*.
2. El sistema muestra la lista de los usuarios con botones de acción.

Asignar roles

Asignar un rol o roles a un usuario en concreto.

Flujo de eventos:

1. El usuario accede a la opción *usuarios*.
2. El sistema muestra la lista de los usuarios con botones de acción.
3. El usuario elige la opción *Asignar rol*.
4. El usuario elige un rol de la lista.
5. El sistema asigna el rol al usuario.

Asignar equipo a entrenador

Asignar equipo o equipos a un usuario con rol entrenador.

Flujo de eventos:

1. El usuario accede a la opción *entrenadores*.
2. El sistema muestra la lista de los usuarios con botones de acción.
3. El usuario elige la opción *Asignar equipo*.
4. El usuario elige un equipo de la lista.
5. El sistema asigna el equipo al entrenador.

Editar información parte pública

Edita información de la parte pública.

Flujo de eventos:

1. El usuario accede a la opción *información web*.
2. El sistema muestra la lista de los campos de información.
3. El usuario elige la opción *Editar*.
4. El sistema actualiza la información.

Añadir avisos a usuarios

Añade avisos a los usuarios dependiendo de su rol.

Flujo de eventos:

1. El usuario accede a la opción *avisos*.
2. El sistema muestra un formulario a rellenar.
3. El usuario rellena el formulario y lo envía.
4. El sistema actualiza los avisos.

Ver temporadas

Ver en sección 6.2.5

Seleccionar temporada de trabajo

Ver en sección 6.2.5

Añadir temporada

Ver en sección 6.2.5

Editar temporada

Ver en sección 6.2.5

Borrar temporada

Ver en sección 6.2.5

Ver categorías

Ver en sección 6.2.5

Añadir categoría

Ver en sección 6.2.5

Editar categoría

Ver en sección 6.2.5

Borrar categoría

Ver en sección 6.2.5

Ver ligas

Ver en sección 6.2.5

Añadir liga

Ver en sección 6.2.5

Editar liga

Ver en sección 6.2.5

Borrar liga

Ver en sección 6.2.5

Ver equipos

Ver en sección 6.2.5

Añadir equipo

Ver en sección 6.2.5

Editar equipo

Ver en sección 6.2.5

Borrar equipo

Ver en sección 6.2.5

Asignar foto a equipo

Ver en sección 6.2.5

Cambiar foto a equipo

Ver en sección 6.2.5

Quitar foto a equipo

Ver en sección 6.2.5

Añadir jornada

Ver en sección 6.2.5

Borrar jornada

Ver en sección 6.2.5

Ver resultados

Ver en sección 6.2.4

Añadir resultados

Ver en sección 6.2.4

Editar resultados

Ver en sección 6.2.4

Ver clasificaciones

Ver en sección 6.2.4

Ordenar clasificaciones

Ver en sección 6.2.4

Salir

Ver en sección 6.2.2

6.3. Estructura del código en el Back-End

Una vez hemos analizado y diseñado, sobre papel, las características de la aplicación; estas se deben pasar a desarrollo. Como se explica anteriormente (ver sección 4.3) la aplicación se desarrolla bajo el patrón MVC. La estructura de directorios del código es la que se muestra en la imagen (Figura 6.8) siguiente:

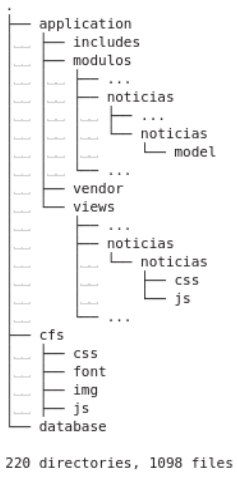

Figura 6.8 Estructura de directorios del Back-End.

6.3.1. application

Es en este directorio donde está casi toda la aplicación. Está divido en otros 4 directorios dependiendo de la función que lleven a cabo los ficheros:

includes

Aquí están los ficheros de configuración y algunas librerías que se necesitan para la aplicación.

modulos

La forma en que se dividen las funciones que se pueden realizar en la aplicación son los módulos. Estos pueden, o no, tener asociado un *modelo* de datos. Es una forma de organizar para un mejor mantenimiento y agregación de nuevas funcionalidades.

A modo de ejemplo se puede ver el módulo *noticias* contenido en *noticias* (Figura 6.8). Dentro de noticias, que no sale en la imagen, también está el módulo *imágenes*.

vendor

Son librerías de apoyo que se actualizan mediante **Composer**[44]. Aquí tenemos el gestor de plantillas **Twig** (ver sección 4.3.3) y el SDK[45] de **Facebook** para utilizar su métodos de autenticación.

views

Es aquí donde se guardan las *vistas* (ver sección 4.3) del patrón MVC. Las vistas, para mejor mantenimiento y corrección de errores, se organizan igual que los módulos a los que pertenecen. Dentro de estas pueden haber otros directorios; como por ejemplo en los que se almacenan CSS (ver sección 4.1.4) y JavaScript (ver sección 4.1.6) específico solo para la vista.

Siguiendo con el ejemplo del módulo *noticias*, este tendrá su vista en la misma ruta dentro del directorio views. Se puede apreciar esto en la imagen (Figura 6.8).

6.3.2. cfs

Se trata del directorio al que apuntará el navegador web y será la entrada del usuario en la aplicación. Es aquí donde está el controlador que será el que sirva las vistas según el módulo que el usuario solicite.

css

Aquí están los ficheros CSS comunes a todas las vistas.

font

Aquí están las fuentes comunes a todas las vistas.

[44] herramienta para la gestión de dependencias en PHP

[45] Software Development Tools. Son herramientas que se dan en formato de paquete para que terceros desarrollen aplicaciones basándose en tu sistema; en este caso Facebook

img

Aquí estás las imágenes comunes a todas las vistas. También es donde el sistema guarda las imágenes que el usuario carga al servidor para luego poder utilizarlas en la aplicación.

js

Aquí están los ficheros JavaScript comunes a todas las vistas. Entre ellos está el código del **CKEditor** (ver sección 4.1.7).

6.3.3. database

Es una copia de seguridad de la base de datos de la aplicación.

6.4. Desarrollo de los módulos

Todos los módulos y sus ficheros han de estar especificados en el fichero **modulos.php** que se encuentra en la ruta *applicacion/includes/modulos.php*. Dentro de este fichero nos encontramos con la variable **$modulos**, que no es más que un array llave - valor como en el siguiente ejemplo:

Código 6.2 Ejemplo de uso de PHP.

```
$modulos = array(
  'noticias'        => 'noticias/noticias/noticias.php',
  ...
)
```

Donde la llave es el nombre del módulo y el valor es la ruta al fichero del módulo.

Esta variable es requerida por el controlador para saber dónde está cada modulo. El controlador de la aplicación se encuentra en *cfs/indexadm.php* y en su código, tras cargar librerías y comprobar permisos, se carga el módulo:

Código 6.3 Ejemplo de uso de PHP.

```
$path_modulo = MODULO_PATH . $modulos[$modulo];
if (file_exists( $path_modulo ))
    include( $path_modulo );
```

6.4.1. Estructura básica de un módulo

Los módulos poseen todos la misma estructura de ficheros y directorios. Esto confiere un mayor control y facilidad a la hora de resolver errores y añadir funcionalidades.

Así, dentro del directorio de un módulo tendremos una serie de ficheros:

1. **modulo.php**: se trata de un fichero con el mismo nombre del módulo que será el fichero principal del módulo.

2. **add.ajax.php**: fichero para agregar un objeto referenciado por el modelo a la base de datos.

3. **edit.ajax.php**: fichero para editar un objeto en la base de datos.

4. **del.ajax.php**: ficher para elimnar un objeto de la base de datos.

Obviamente, pueden existir más ficheros que estos. La lista que hemos visto se considera básica, aunque no es necesario que estén todos.

Dentro del directorio del *módulo* tenemos un directorio llamado **model**. Es aquí donde especificamos el modelo de datos que vamos a utilizar y su repositorio de operaciones. Así, dentro de este directorio tendremos, mínimo, dos ficheros:

1. **Objeto.php**: descripción de atributos y métodos del objeto del modelo.

2. **ObjetoRepository.php**: repositorio de acciones, mayoritariamente contra la base de datos, utilizando objetos del tipo Objeto.

La organización de las vistas ya han sido explicadas en la sección 6.3.1.

6.4.2. Módulo home

Se trata del módulo que se ejecuta cuando se entra en la URL de la aplicación. Si no se está logueado se da la opción de que el usuario lo haga. Ver imagen (Figura 6.9) siguiente:

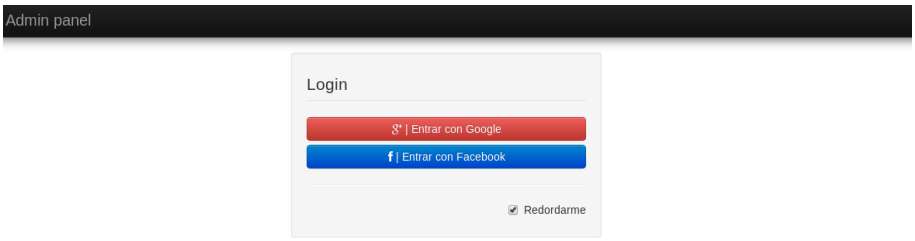

Figura 6.9 Vista home.

Una vez el usuario haga login, será el módulo *login* el encargado de recuperar los roles y loguear al usuario.

6.4.3. Módulo temporadas

Si el usuario tiene permisos para ello, una vez logueado en el sistema, podrá acceder a los diferentes módulos de la aplicación. Entre ellos se encuentra el módulo *temporadas*. Podemos ver una captura de la vista en la imagen (Figura 6.10). Desde este módulo se gestionan las temporadas. Se divide en varias acciones. A las básicas vistas en 6.4.1 se le añade:

asign.ajax.php: asigna la temporada de trabajo.

Figura 6.10 Vista temporadas.

6.4.4. Módulo categorias

Este módulo posee las acciones básicas vistas en 6.4.1. Se puede ver una captura de su vista en la imagen (Figura 6.11) siguiente:

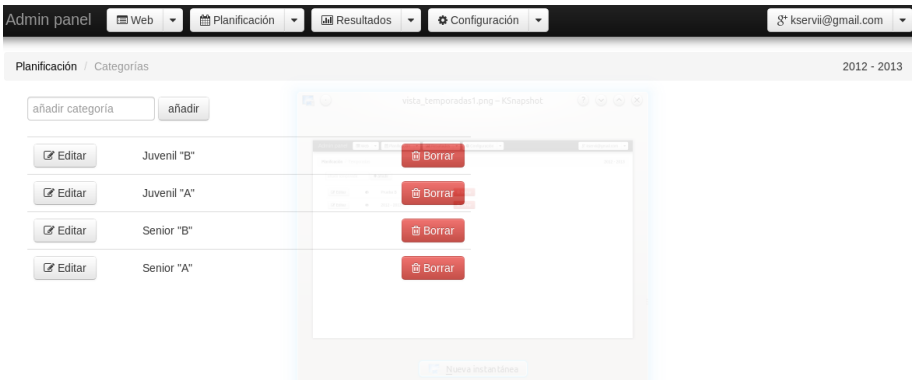

Figura 6.11 Vista categorias.

6.4.5. Módulo ligas

El módulo posee todas las acciones referente a la gestión de ligas de la aplicación. A las acciones básicas vistas en 6.4.1 se le añaden:

datos_liga.php: carga asíncronamente los datos de una liga cuando se quiere editar la misma.

modifica_orden.php: guarda la nueva posición dentro de la lista que se le asigna a la liga.

mostrar_liga.php: cambia el valor de mostra/ocultar una liga en la parte pública.

tabla_ligas.ajax.php: carga la lista de ligas tras agregar una nueva liga.

Una captura de la vista principal la podemos ver en la imagen (Figura 6.12) siguiente:

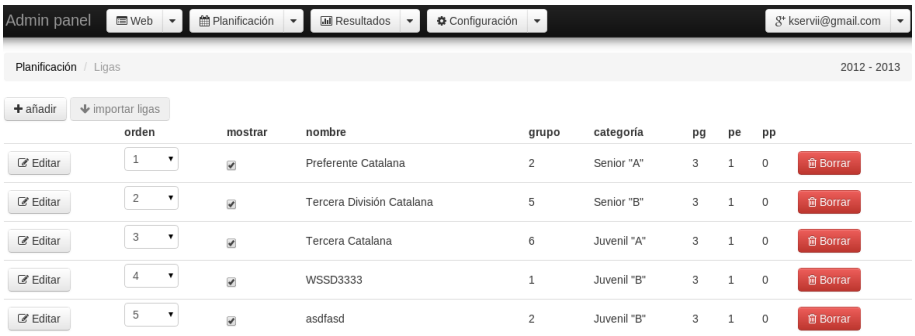

Figura 6.12 Vista ligas.

Cuando el usuario quiere agregar una nueva liga o editar una ya existente, aparece una ventana modal para poder llevar a cabo dicha acción. Vemos una captura de dicha ventana en la imagen (Figura 6.13 siguiente:

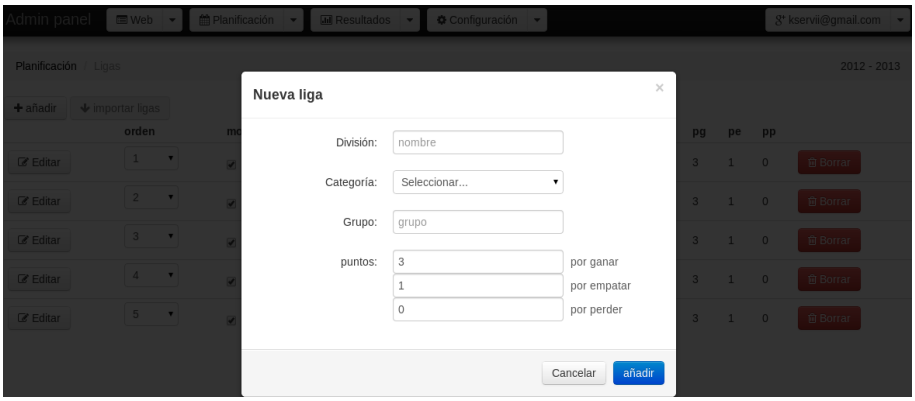

Figura 6.13 Vista venta modal añadir/editar ligas.

La diferencia entre la ventana de añadir y guardar, es que en esta última se cargan los datos de la liga a modificar y la opción del botón se denomina *editar*.

6.4.6. Módulo equipos

Este módulo se encarga de la gestión de los equipos. A las acciones básicas vistas en 6.4.1 se le añaden:

asign.foto.php: asigna una foto a un equipo.

comprobar_liga_iniciada.php: comprueba si la liga ya ha sido iniciada.

datos_equipo.php: recupera los datos del equipo para su edición.

equipo_row.php: renderiza la vista de una sola fila de la tabla de equipos.

equipos_liga.php: muestra la lista de equipos una vez hemos elegido la liga en la página principal.

La captura de la vista principal la podemos ver en la imagen (Figura 6.14) siguiente:

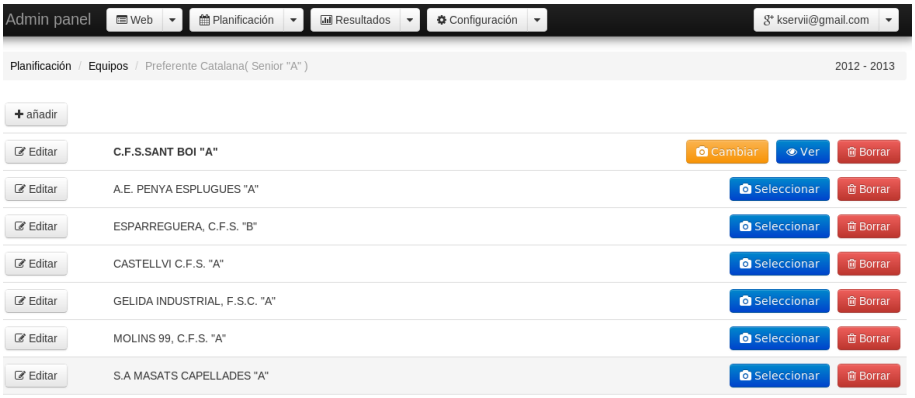

Figura 6.14 Vista equipos.

Cuando el usuario quiere agregar un equipo o editar uno existente, aparece una ventana modal para poder llevar a cabo la acción. Podemos ver una captura de la ventana en la imagen (Figura 6.15) siguiente:

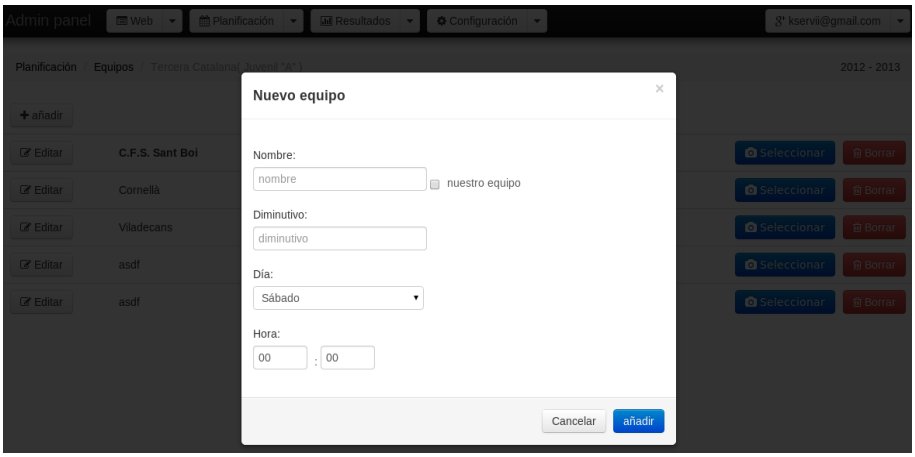

Figura 6.15 Vista venta modal añadir/editar equipos.

La diferencia entre la ventana de añadir y guardar, es que en esta última se cargan los datos de la liga a modificar y la opción del botón se denomina *editar*.

6.4.7. Módulo jornadas

Este módulo se encarga de la confección del calendario de cada liga. Es un módulo un poco especial y la lista de ficheros para las acciones es particular:

jornadas_liga.php: muestra la lista de equipos (a la izquierda) y conforme se van eligiendo genera los enfrentamientos (a la derecha).

listar_equiposJSON.php: se encarga de consultar a la base de datos por los equipos de la liga elegida y devolverlos en formato JSON.

comprueba_posibles.php: hace una comprobación, una vez selecciona un equipo de la lista, de los posibles enfrentamientos que podría tener eliminando los equipos contra los que ya ha sido emparejado.

guardar_jornada.php: guarda la jornada con todos los enfrentamientos en la base de datos.

La captura de la vista la podemos ver en la imagen (Figura 6.16) siguiente:

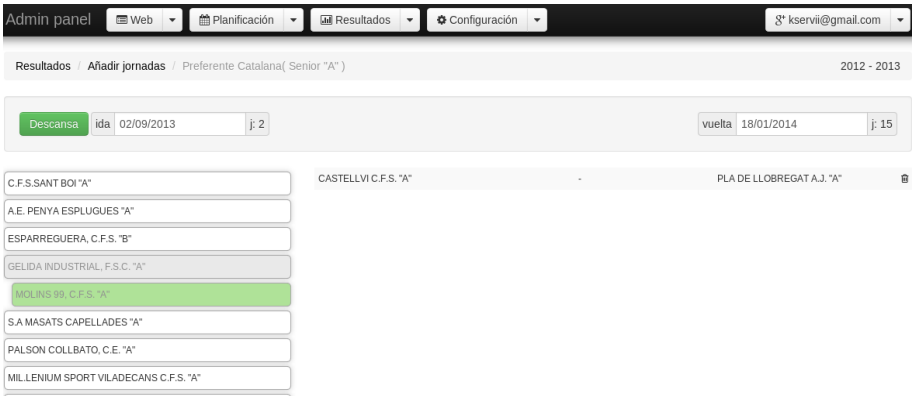

Figura 6.16 Vista jornadas.

6.4.8. Módulo resultados

Este módulo se encarga de la introducción y edición de los resultados. En este caso, al igual que el módulo *jornadas*, se trata de un módulo algo especial y su estructura no se parece en nada en la vista en 6.4.1:

resultados_liga.php: una vez elegida la liga, muestra la última jornada que falta por introducir los resultados.

actualizar_resultados.php: cuando los resultados ya han sido metidos pero el usuario quiere modificar algún resultados, es este fichero el que se encarga de guardar los cambios.

guardar_resultados.php. una vez el usuario ha añadido los resultados de la jornada, se encarga de guardarlos en la base de datos.

La captura de la vista la podemos ver en la imagen (Figura 6.17) siguiente:

Figura 6.17 Vista resultados.

6.4.9. Módulo clasificaciones

Este módulo se encarga de la visualización y modificación de las clasificaciones. La única función que posee es:

clasificacion_liga.php: tras elegir la liga de la que queremos obtener la clasificación, muestra la clasificación de la última jornada con los resultados introducidos.

actualizar_clasificaciones.php: una vez cambiamos un equipo de orden en la clasificación se encarga de guardarlo, de forma asíncrona, en la base de datos.

La captura de la vista la podemos ver en la imagen (Figura 6.18) siguiente:

Figura 6.18 Vista clasificaciones.

6.4.10. Módulo imagenes

El módulo se encarga de al carga y eliminación de imágenes. La lista de acciones que posee es (solo se pondrán las que no estén listadas en 6.4.1):

ckeditor.picker.php: se trata de un selector de imagen que acoplamos al editor CKEditor (ver sección 4.1.7) para poder añadir imágenes a las noticias.

image.picker.php: se trata de un selector más genérico para poderlo llamar desde otros módulos de la aplicación. En este caso se llama desde el módulo equipos para asignar una foto a un equipo.

upload.ajax.php: carga, asíncronamente, la imagen que el usuario haya elegido y recarga la vista para que aparezca.

La captura de la vista la podemos ver en la imagen (Figura 6.19) siguiente:

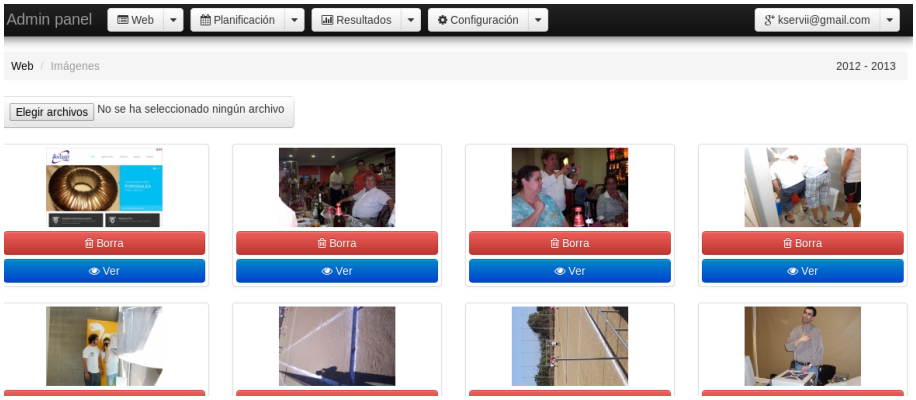

Figura 6.19 Vista imagenes.

6.4.11. Módulo noticias

Este módulo se encarga de la gestión de las noticias. Las acciones posibles siguen el esquema de la lista lista básica (ver 6.4.1).

La captura de la vista, en que se ve el lista de las noticias con las posibles acciones, la podemos ver en la imagen (Figura 6.20) siguiente:

Figura 6.20 Vista noticias.

Cuando un usuario quiere añadir o editar una noticia, el sistema cambia a una nueva página donde se carga el editor. Esta página la podemos ver en la imagen (Figura 6.21) siguiente:

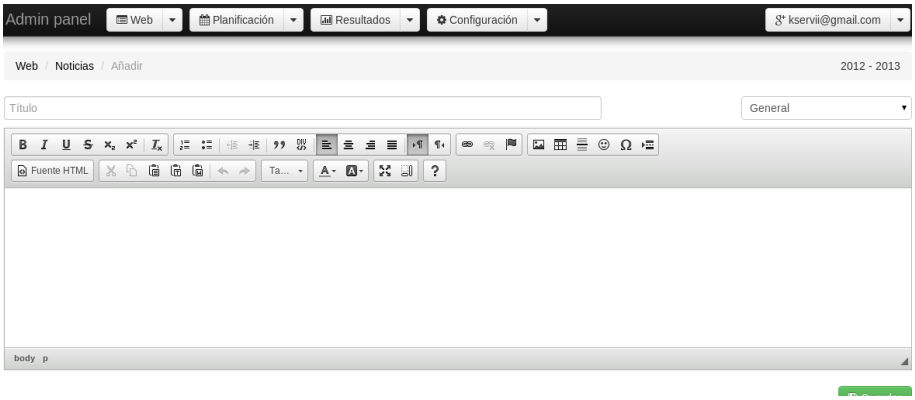

Figura 6.21 Vista página para añadir/editar noticia.

6.4.12. Módulo usuarios

Este módulo se encarga de la gestión de los usuarios. Dado que se utilizan los sistemas de Facebook y Google para el acceso, no se pueden añadir usuarios; eso lo hace el sistema automáticamente. Lo que si se puede hacer son las siguientes acciones:

del.ajax.php: borrar un usuario.

change_rol.php: cambiar el rol a un usuario.

La captura de la vista la podemos ver en la imagen (Figura 6.22) siguiente:

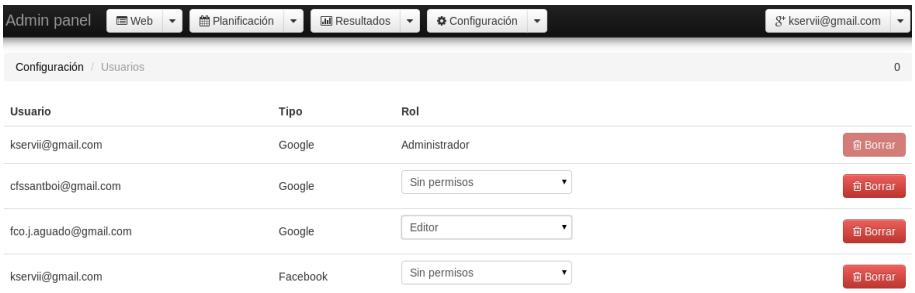

Figura 6.22 Vista usuarios.

6.4.13. Módulo entrenadores

Este módulo se encarga de la gestión del equipo que se le asigna a cada entrenador. Solo se encarga de una simple función:

asign_liga.php: asigna una liga al entrenador.

La captura de la vista la podemos ver en la imagen (Figura 6.23) siguiente:

Figura 6.23 Vista entrenadores.

6.4.14. Módulo notificaciones

Este módulo se encarga de la gestión de las notificaciones que los administradores envían a los usuarios. De la lista básica vista en 6.4.1) solo realiza la opción de *añadir* y *borrar*; dado que no se creen oportunas más acciones.

La captura de la vista la podemos ver en la imagen (Figura 6.24) siguiente:

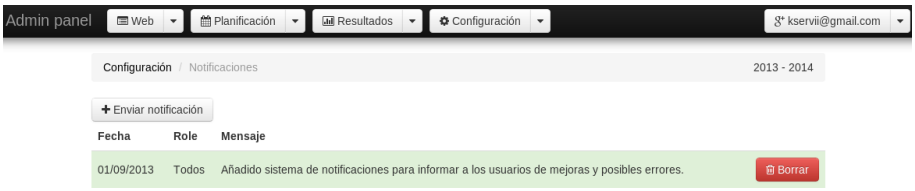

Figura 6.24 Vista notificaciones.

Cuando un administrador quiere añadir una nueva notificación, el sistema muestra una ventana modal que podemos ver en la imagen (Figura 6.25) siguiente:

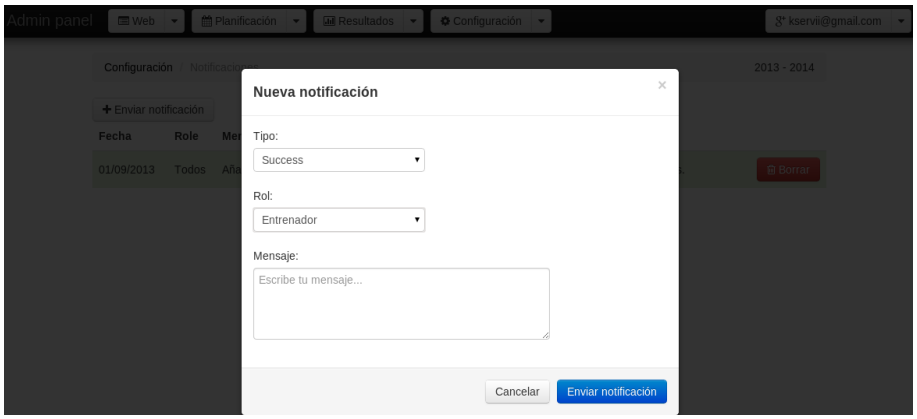

Figura 6.25 Ventana modal para añadir notificaciones.

6.5. Zona pública

Un gran número de aplicaciones web están orientadas a la visualización pública; este caso no es diferente. Todo el desarrollo de la zona de administración es utilizado para nutrir de información a la zona pública. En la imagen siguiente (Figura 6.26), vemos una captura; esta será la que vean nuestros seguidores, socios y demás usuarios que lleguen a nuestra web.

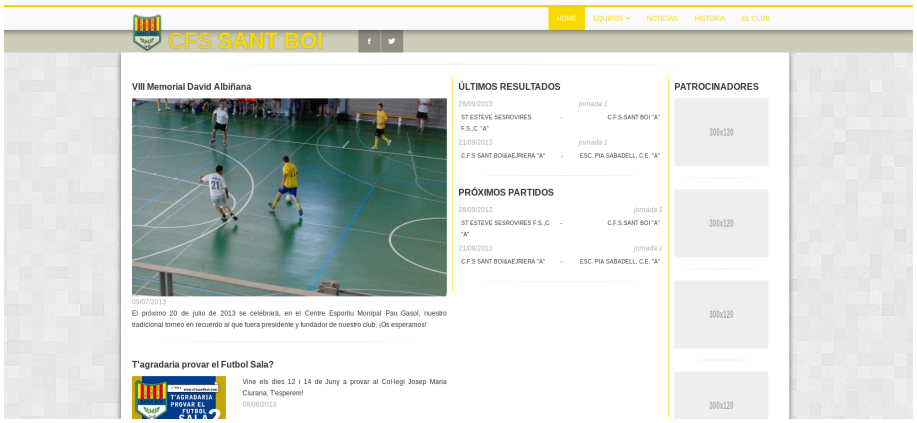

Figura 6.26 Vista pública.

Dado la falta de diseñador, y la importancia del mismo hoy en día, se ha decidido basar el diseño de la zona pública en una plantilla para el framework CSS utilizado. La plantilla, en su forma original, se puede ver en la imagen (Figura 6.27) siguiente.

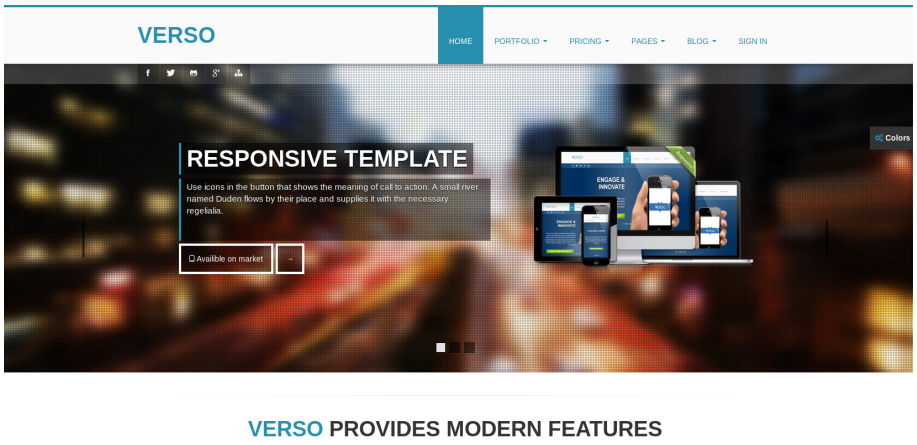

Figura 6.27 Vista de la plantilla original.

Como se puede apreciar se ha modificado bastante; añadiendo unos elementos y quitando otros que se consideran prescindibles para el tipo de información que se quiere ofrecer.

El diseño es bastante tradicional. Dispone de una barra de navegación superior donde el usuario puede acceder a los enlaces más relevantes de nuestra web.

Al entrar en la zona pública, a simple vista el usuario podrá obtener las últimas noticias, últimos resultados, próximos partidos y una lista de patrocinadores.

El diseño ha sido realizado bajo la premisa Responsive web design (o diseño de web adaptativo[46]) utilizando la herramienta **Vista de diseño adaptable** del navegador utilizado para el desarrollo: Mozila Firefox. Así, se ha tenido que decidir qué información es más relevante para algunas vistas más pequeñas.

Por ejemplo, en el caso de la vista para smartphones, se decide eliminar últimos resultados y próximos partidos en detrimento de las noticias (para estar bien informado) y los patrocinadores (si pagan han de aparecer en primera página). Podemos ver una captura de la herramienta en funcionamiento en la imagen (Figura 6.28) siguiente.

[46] Tipo de diseño web que se adapta a todo tipo de dispositivos dependiendo de su resolución

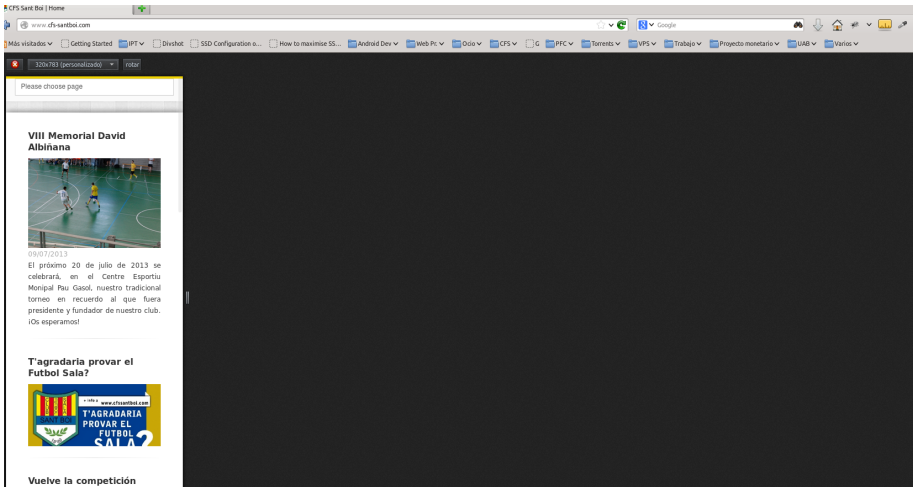

Figura 6.28 Vista para smartphones.

En este caso, gracias al framework utilizado, el menú superior pasa a transformarse en un **select**[47] que nos permite una navegación mejor en este tipo de dispositivos.

En la vista para tabletas en vertical se decide el mismo tipo de contenido que para smartphones. La diferencia es que, mientras en smartphones los patrocinadores salen debajo de las noticias; en las tabletas, al tener más espacio, estos aparecen al lado. Se puede ver cómo queda en la imagen (Figura 6.29) siguiente.

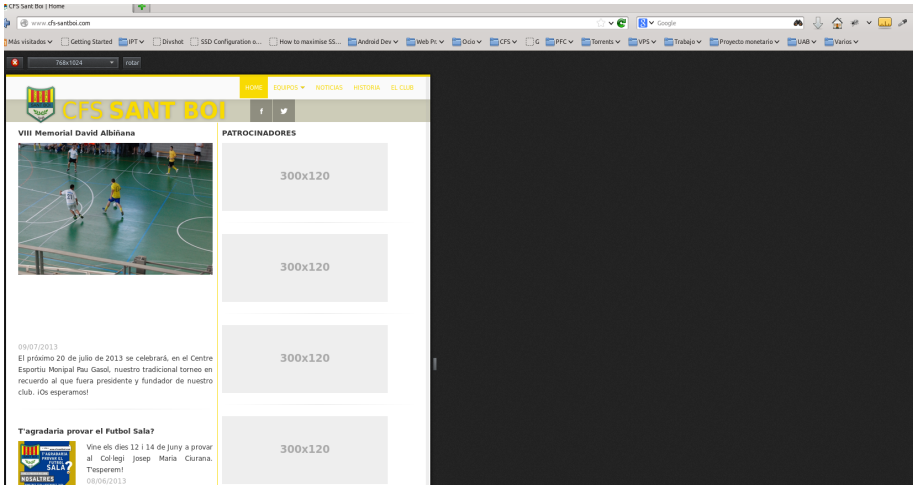

Figura 6.29 Vista para tabletas.

[47] Elemento HTML para la selección de un elemento de una lista

7. Pruebas

"¿Cómo debería saber si funciona? Para eso están los Beta testers. Yo solo programo." (Linus Torvalds)

7.1. Introducción

Las pruebas de software son un conjunto de técnicas que permiten encontrar errores de una aplicación y facilita, a su vez, conseguir un correcto funcionamiento.

Estas no garantizan la ausencia de errores, sino que nos ayudan a encontrar esos que no se han tenido en cuenta durante el desarrollo; ya sea por error de análisis, en la programación o en la posterior puesta en marcha en producción.

Se han realizado tres tipos de prueba básicos y comunes.

7.1.1. Pruebas unitarias

Con estas pruebas se intenta comprobar el correcto funcionamiento de un módulo de código sin tener en cuenta el conjunto.

Estas pruebas se han aplicado a los formularios y recogida de sus datos, así como a otras funcionalidades como: cálculo de clasificaciones, cálculo de resultados, generador de resúmenes de texto, generador de slugs, etc.

7.1.2. Pruebas de integración

Con estas pruebas se intenta comprobar el correcto funcionamiento del conjunto o subconjunto examinado.

Este tipo de prueba se han llevado a cabo durante todo el desarrollo del proyecto. Por ejemplo, la generación de una nueva temporada conlleva una cadena de acciones que relaciona varios módulos: crear temporada, crear categorías, crear una liga vinculada a una categoría y crear equipos vinculados a la liga. Con esta prueba se asegura que la cadena funciona en todos sus pasos.

7.1.3. Pruebas de compatibilidad

En el caso de las aplicaciones web se ha de comprobar el buen funcionamiento en los navegadores más comunes.

A día de hoy, se puede afirmar que las pruebas han de realizarse obligatoriamente en los siguientes 4 navegadores de escritorio: Microsoft Internet Explorer, Apple Safari, Mozilla Firefox y Google Chrome. Para las pruebas en los navegadores de Microsoft y Apple se ha optado por la utilización de virtualización.

Zona de administración

A continuación tenemos una captura de la zona de administración bajo Internet Explorer en su versión 10 (Figura 7.1). No se han detectado errores derivados de la utilización de dicho navegador.

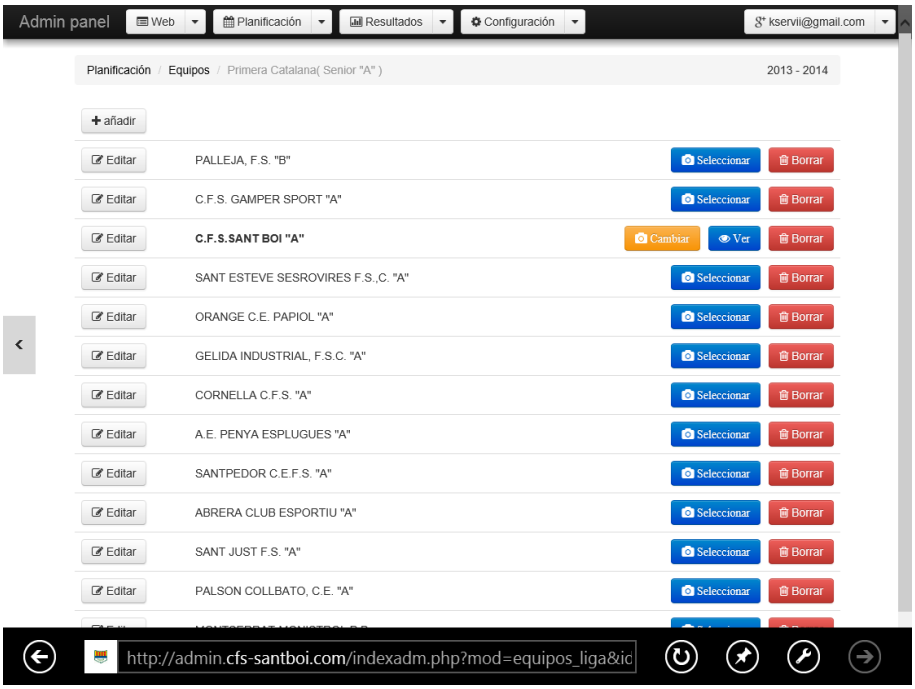

Figura 7.1 Zona de administración en Internet Explorer 10.

En las pruebas para Apple Safari, en la versión 5.1.7 del navegador, se aprecia un pequeño desajuste en los gráficos del menú superior. No siendo crítico y siendo la parte de administración, donde la funcionalidad ha de privar frente al diseño, se opta por apuntarlo en líneas futuras. Se puede ver una captura a continuación (Figura 7.2).

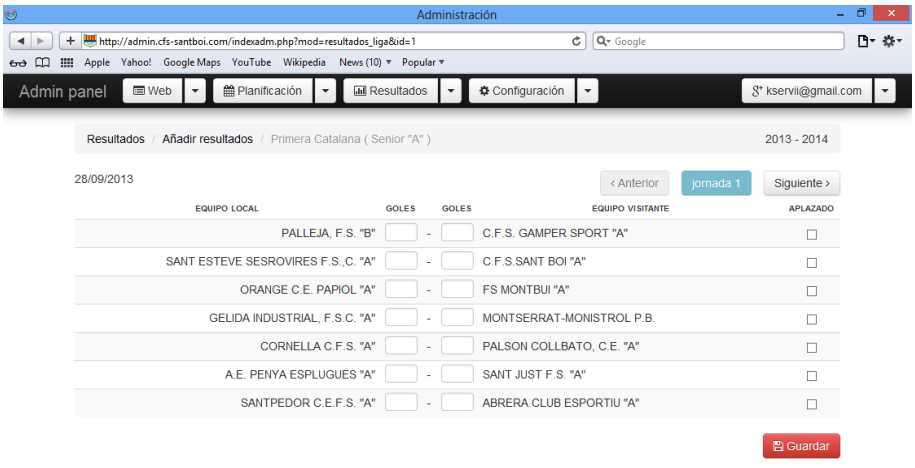

Figura 7.2 Zona de administración en Apple Safari 5.1.7.

Las pruebas para Google Chrome, en la versión 29.0 del navegador, han resultado satisfactorias. En la siguiente captura se puede ver cómo se visualiza (Figura 7.3).

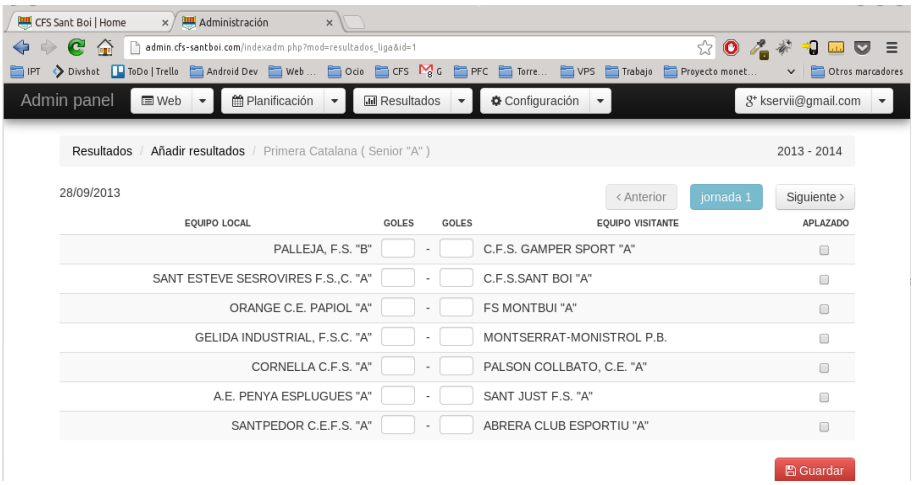

Figura 7.3 Zona de administración en Google Chrome 29.0.

En el caso de Firefox, y debido a que ha sido el navegador utilizado durante todo el desarrollo, podemos asegurar que el 100 % de las funcionalidades desarrolladas funcionan en dicho navegador. A continuación se puede observar una captura de pantalla para dicho navegador en su versión 23 (Figura 7.4).

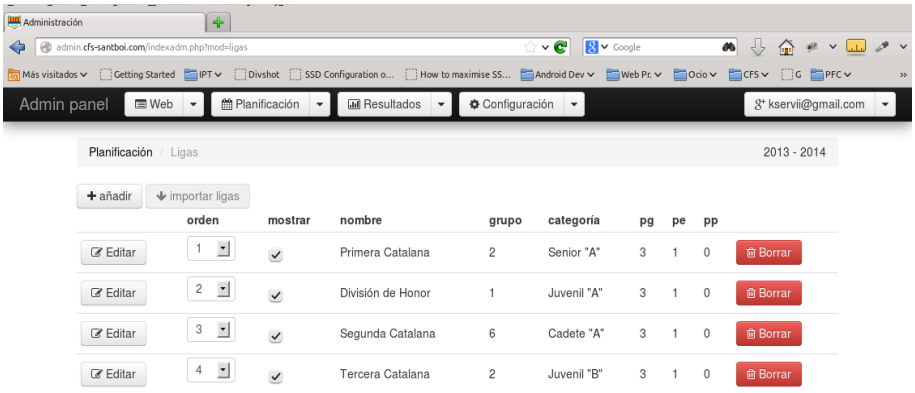

Figura 7.4 Zona de administración en Mozilla Firefox 23.

La zona de administración no ha sido diseñada bajo la premisa Responsive web design (o diseño de web adaptativo). Esto es así dado que se ve oportuna la utilización de un PC para el correcto funcionamiento. No así, como ya se vio en el apartado 6.5, la zona pública si ha sido diseñada de este modo.

Zona pública

Para las pruebas de compatibilidad de la zona pública se vuelven a utilizar los mismos navegadores que para la zona de administración; añadiendo la utilidad de Mozilla Firefox **Vista de diseño adaptable**. Este tipo de vista nos permite ir cambiando entre las diferentes resoluciones que tenemos hoy día para ver qué tal se vería tanto en tabletas como en smartphones.

Para las versiones de escritorio de los navegadores no se han detectado errores de visualización.

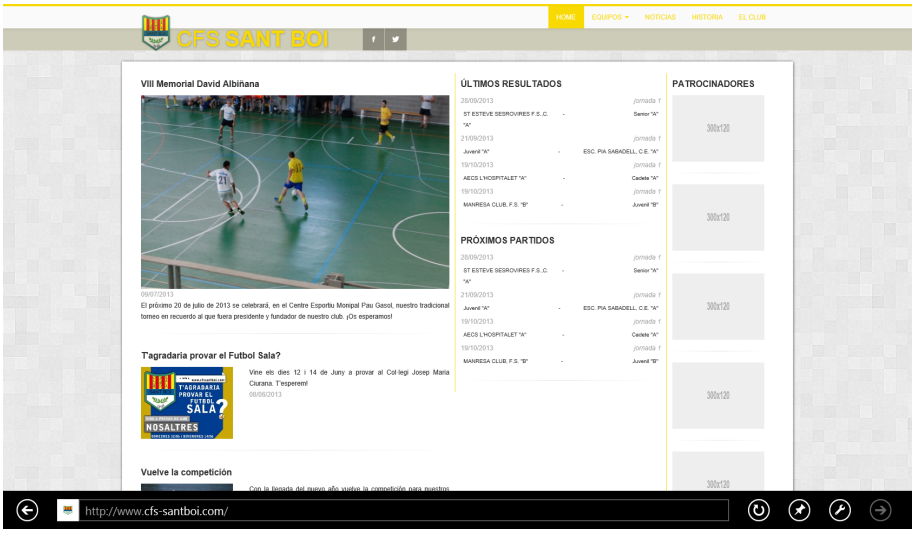

Figura 7.5 Zona pública en Internet Explorer 10.

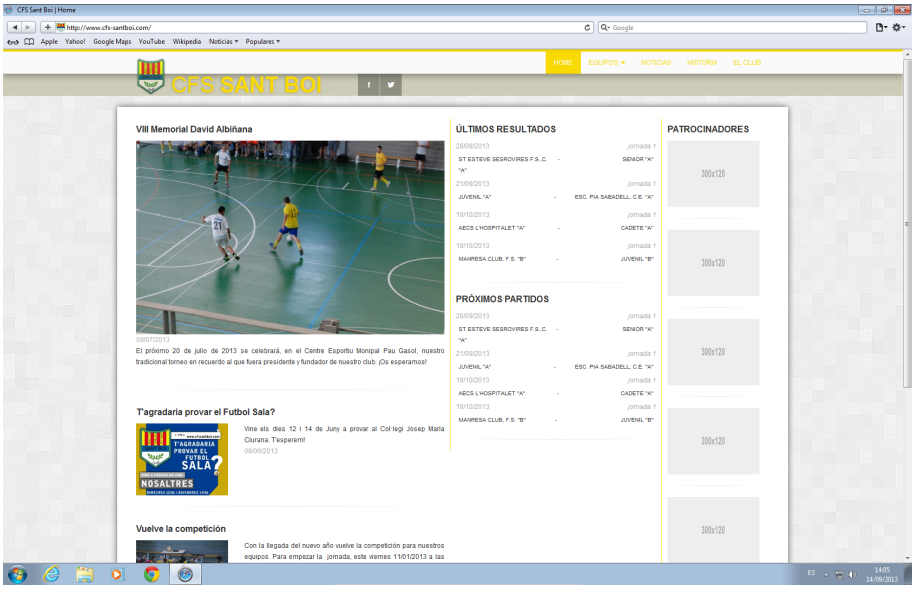

Figura 7.6 Zona pública en Apple Safari 5.1.7.

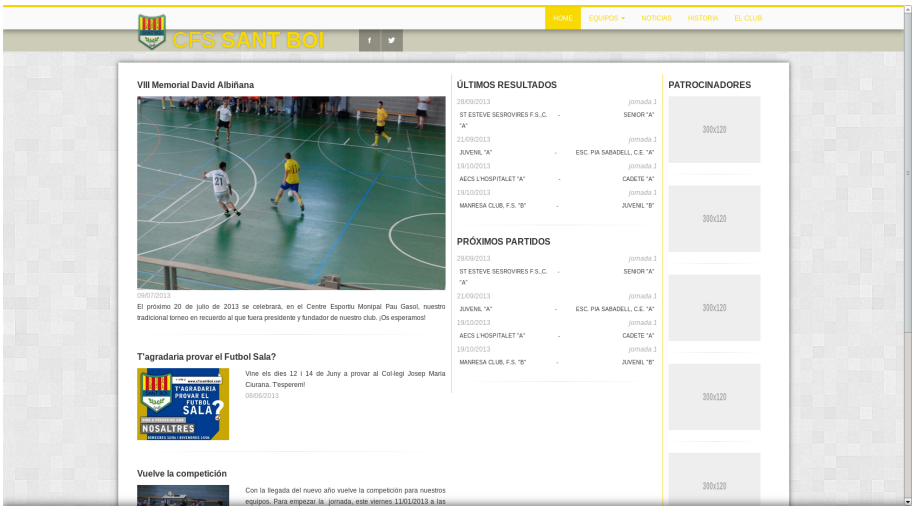

Figura 7.7 Zona pública en Google Chrome 29.0.

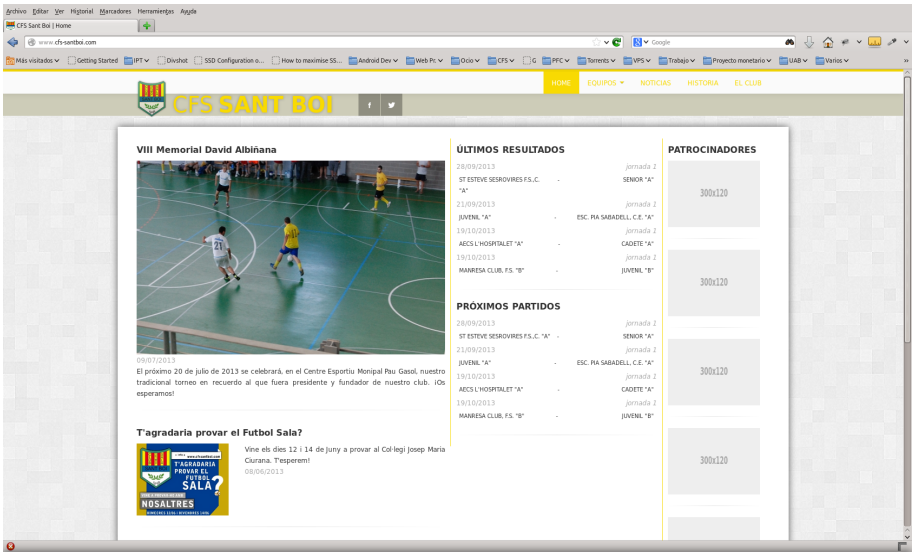

Figura 7.8 Zona pública en Mozilla Firefox 23.

Para las versiones para smartphones y tabletas hubieron algunos errores de maquetación que fueron resueltos en esta fase del proyecto. Las capturas de la visualización en este tipo de dispositivos podemos verlas en la sección 6.5: en la Figura 6.28 para smartphones o en la Figura 6.28 para tabletas.

7.1.4. Errores detectados

Durante el proceso de prueba se han encontrado errores que ya han sido solucionados.

Algunos de los errores, típicos de aplicaciones con formularios, que se han encontrado:

1. Formularios con ID mal escritas y que en el destino se recogen datos erróneos.
2. Rutas de destino incorrectas.
3. Comprobaciones previas mal planteadas y se producen fallos al añadir y/o editar.
4. Al vaciar la base de datos y empezar de cero, la aplicación fallaba dado que no encontraba ninguna temporada.
5. Errores de maquetación para dispositivos móviles.

Por último, a la hora de pasar a producción se detectaron ciertos problemas que no se habían tenido en cuenta antes. Dichos problemas surgieron dado que el entorno de desarrollo no era 100 % idéntico al de producción.

1. A la hora de subir fotografías se produce error dado que el límite del fichero a subir está fijado en 2 MB.
2. Se produce el error "Strict Standards only should be passed by reference" debido al cambio de versión de PHP. Se desactivan los avisos por no considerarlo un error fatídico a la espera de una revisión de la versión que lo corrija.

8. Conclusiones

Para que el siguiente proyecto que emprendamos salga mejor que el anterior, al finalizar un proyecto hay que echar la vista atrás y valorar todo el conjunto.

El proyecto partía de una aplicación hecha que no era nada mantenible y con una gestión de usuarios y permisos inexistente. De este hecho surge la necesidad de la creación, desde cero, de una nueva aplicación que solventara las principales deficiencias anteriores.

El primer punto de los objetivos: dotar a la aplicación de una gestión de usuarios, ha sido una experiencia complicada. Hacer un sistema de permisos robusto, pero que no llevase mucho tiempo para no quitárselo a otros objetivos importantes del proyecto. En este caso, haber separado el código en módulos ha dotado de la sencillez necesaria para poder alcanzar el objetivo deseado sin demorar demasiado.

El siguiente objetivo: la columna vertebral del proyecto. Se trataba de hacer un gestor de competición de liga para que los miembros de la junta pudieran gestionar los equipos. El objetivo se ha alcanzado con creces. La gestión es la más completa que se ha podido implementar y permite hacer todo lo que se quería: temporadas, categorías, ligas, equipos, jornadas, resultados, partidos aplazados, resultados parciales... Todo lo deseado se ha podido realizar.

El último de los objetivos, pero no por ello el menos importante, es la generación de noticias. Era un objetivo sencillo de alcanzar, y se ha conseguido. Se pueden añadir noticias y estas se ven tanto en la portada como en la sección de noticias de la parte pública; con sus respectivos botones de redes sociales.

Viendo el conjunto de todo el proyecto, se puede afirmar que los objetivos han sido alcanzados completamente.

Experiencia personal

Como proyecto final de carrera me ha permitido aplicar conceptos que hacía tiempo que deseaba poner en práctica. Ayudado con los conocimientos adquiridos durante años en la carrera, he podido crear desde cero una aplicación Modelo-Vista-Controlador.

La parte más dura, dado que carezco de las capacidades necesarias, ha sido el diseño de la zona pública. Me ha hecho sufrir y pensar muchísimo. Por ello es que me siento orgulloso del resultado y esperando entregar el proyecto para poder aplicar las modificaciones que se me han ido ocurriendo durante la fase de desarrollo y planteo en la sección 9.

Punto a parte merece la utilización de LaTeXcomo herramienta para la creación de esta memoria. Al principio me sentía algo reacio a utilizarla; finalmente ha resultado bastante más ameno que haber utilizado una herramienta estilo WYSIWYG[48] como por ejemplo: Microsoft Word. Lo achaco a que ha sido como programar; maquetando y cuadrando los textos con las imágenes, tablas y listas.

[48] What You See Is What You Get o lo que escribes es lo que obtienes

9. Líneas futuras

Porque, por mucho que nos guste el pasado o el presente, hay que seguir mirando hacia el futuro e ir creciendo con el tiempo. Es por ello que, incluso antes de terminar un proyecto, ya empezamos a pensar en mejoras para el mismo.

Una de las ventajas de saber diseñar una aplicación tras un exhaustivo análisis, es la facilidad con la que se pueden añadir mejoras o características con un menor esfuerzo. Durante la fase de desarrollo han surgido características extra que, de haber querido implementarlas durante la misma fase se hubiera tardado demasiado en terminar el proyecto.

1. **Una gestión completa de jugadores**: se trata de una asignatura pendiente, por así decirlo. La posibilidad de poder añadir jugadores a un equipo y poder gestionar sus estadísticas.
2. **Gestión de partidos amistosos**: dado que la gestión de competiciones de liga ya está hecha, no se tardaría demasiado en desarrollarse un sistema para plantear partidos amistosos.
3. **Creación de calendario**; la creación de un calendario donde salieran todos los partidos del club y poder avanzar y retroceder para ver el día y la hora.
4. **Gestión de torneos**: todos los veranos el club organiza un torneo en recuerdo al que fuera nuestro fundador. Un gestor de torneos facilitaría muchas tareas y así los jugadores participantes podrían consultar la información en la web durante el torneo.
5. **Comparte tu foto**: se trataría de una sección para que los aficionados pudieran subir fotos e incluso, si están registrados, poder poner cómo va el resultado en directo desde el campo.
6. **Tienda del club**: para poder vender prendas del club, a modo merchandising. Es algo que se me ha reclamado este verano y que se planteará hacer en un futuro.
7. **Gestor de patrocinadores**: posibilidad de poder modificar los patrocinadores y decidir en qué orden aparecen sin la necesidad de modificar el código a mano.
8. **Obtención de información de FCF**: posibilidad de obtener información (escudo, lugar de partidos, resultados...) directamente de la página de la Federación Catalana de Fútbol sin intervención humana manual.

10. Bibliografía y Referencias

Todo proyecto requiere una documentación antes y, sobretodo en el caso de proyectos de software, durante. Es importante recordar que nadie nace aprendido y que hay que apoyarse en lo que antes ya se ha hecho; hay que saber aprovechar los recursos a nuestro alcance y eso, hoy en día, es casi infinito: internet.

10.1. Bibliografía

Dado que las consultas para las siguientes páginas han sido continúas, se pondrá la fecha de la última consulta según el historial del navegador web.

The PHP Group. (2013). PHP: Hypertext Preprocessor.
Consultada el 24 de agosto de 2013, en http://php.net

Stack Eexchange inc. (2013). Stack Overflow.
Consultada el 20 de agosto de 2013, en http://stackoverflow.com

The jQuery Foundation. (2013). JQuery API Documentation.
Consultada el 26 de agosto de 2013, en http://api.jquery.com

The jQuery Foundation. (2013). JQuery UI API Documentation.
Consultada el 26 de agosto de 2013, en http://api.jqueryui.com

GitHub, Inc. (2013). GitHub.
Consultada el 26 de agosto de 2013, en https://github.com

Twitter. (2013). Bootstrap.
Consultada el 28 de agosto de 2013, en http://getbootstrap.com/2.3.2/

Maks Surguy. (2012). Gallery of free HTML snippets for Twitter Bootstrap.
Consultada el 12 de junio de 2013, en http://bootsnipp.com/

Alexis Sellier. (2013). LESS The Dynamic Stylesheet language.
Consultada el 13 de junio de 2013, en http://lesscss.org/

Sensio Labs Network. (2013). Twig - The flexible, fast and secure PHP template engine.
Consultada el 7 de agosto, en http://twig.sensiolabs.org/

Nacho Pacheco. (2013). Manual de Twig en Español.
Consultada el 7 de agosto, en http://gitnacho.github.io/Twig/

10.2. Referencias

Agencia Española de Protección de Datos (2012)
Consultada el 23 de octubre de 2012, en https://www.agpd.es

Normativa de proyectos de ingeniería técnica (2012)
Consultada el 15 de septiembre de 2012, en http://www.uab.cat