UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA DE TELECOMUNICACIÓN: SISTEMAS Y REDES DE TELECOMUNICACIONES

PROYECTO FIN DE CARRERA

VIRTUALIZACIÓN DE UNA APLICACIÓN WEB PARA SU USO CON CONTENEDORES DOCKER

Autor: Darío Dosantos
Tutor: Andrés Marín

21 de septiembre de 2017

Agradecimientos

Gracias a Juan por apoyarme.
Gracias a Fede, Raúl, Oli y Baca por responderme.
Gracias a Jesús por sugerirme.
Gracias a mis padres por...bueno, por todo.
Gracias a Andrés por tolerarme.
Dedicado a mis abuelas Elisa y Remedios, y a mi abuelo Luis. Perdonad la tardanza.

Resumen

Este Proyecto Fin de Carrera plantea el ciclo de vida completo de una aplicación web de gestión orientada a llevar los datos estadísticos de un equipo amateur de baloncesto. Consiste en el desarrollo de la aplicación en sí, su adaptación a nuevas versiones de software, su integración con aplicaciones de terceros (Facebook en este caso concreto), su securización mediante certificados y su despliegue en un entorno estable y accesible de una manera rápida y sencilla, haciendo uso de tecnologías de virtualización y paquetización (Docker).
La aplicación permite identificación de los usuarios, presenta una variedad de información en diferentes tablas y permite al administrador operar sobre los datos del sistema, añadiendo, editando o eliminando, según la necesidad. También se integra con la red social Facebook para facilitar el acceso a usuarios no registrados previamente y está desplegada mediante el uso de la tecnología de Docker, que hace los despliegues sencillos y cómodos, optimizando los recursos del sistema que la aloje.

El presente documento abarca desde la fase de análisis de requisitos hasta el despliegue en un servidor final, explicando cada fase del proyecto y cada decisión tomada.

Abstract

The present PFC (*Proyecto Final de Carrera*) propounds the full life cicle of a web application, oriented to the management of the statistic data of an amateur basketball team. It consists on the development of the code, adapting it to fresh versions of the software, the integration with third party applications (Facebook, in this particular case), the securization of the system by using certificates and the deployment into a stable environment in a easy way, making use of virtualization and packaging technologies (Docker).
The application identifies different users and roles, shows the information presenting a variety of tables and allows the administrator to operate over the system data, offering the possibility to add, edit or remove data according to the needs of the moment. It also integrates with the social network Facebook in order to grant access to users who were not previously registered, and it is deployed by means of Docker technology, that makes the deployments easy and optimizes the resources of the system that hosts it.

The span of this document includes the full scope of the project, from the requirement analysis step to the final deployment in a secure server. It also explains each step and decission.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Motivación del proyecto

Uno de los sectores que ha experimentado un mayor crecimiento durante las últimas décadas ha sido el desarrollo web. Cada vez más aspectos de la realidad han ido encontrando acomodo en Internet modificando los hábitos de la población, desde aspectos profesionales a personales, de ocio, salud, etc. Las funcionalidades que ofrecen las aplicaciones web se han ido ampliando con el paso del tiempo, desde aquellos inicios en los que páginas web estáticas, escritas por completo en HTML añadiendo tal vez imágenes sin demasiado tratamiento, eran cargadas mediante conexiones de 54 kbps. Hoy en día una gran variedad de tareas antes impensables se llevan a cabo a través de la web, tales como compras, movimientos bancarios, acceso a la información o a contenidos multimedia, juegos, etc. También la forma de trabajar ha variado sustancialmente al tener acceso a compartición inmediata de archivos, comunicaciones a grandes distancias, herramientas de organización y gestión remotas y demás.

Y por supuesto, en el plano personal, ha abierto la puerta al gran público a multitud de posibilidades antes lejanas. Hoy cualquier usuario medio puede tener un espacio de comunicación personal a través de un blog, administrar sus cuentas con su banco on-line, promocionar su empresa mediante páginas web a medida o diversificar su ocio con multitud de diferentes oportunidades, por no hablar del tremendo auge de redes sociales como Twitter o Facebook que han modificado las relaciones interpersonales y los hábitos de consumo del público en general. De esta forma, podemos plantear hoy actividades que antes no tendrían demasiado sentido, por ejemplo llevar la administración de un hobbie tan usual y extendido como un equipo deportivo amateur.

Lo que en el pasado sería una idea difícil de llevar a cabo, hoy es una

tarea asumible de realizar: con una interfaz web usable se puede tener al equipo conectado; con las estadísticas publicadas, relacionadas, instantáneas.

Esa es la justificación del presente proyecto, puesto que un profesional con formación en el ámbito de la telemática y el desarrollo web puede, de manera más o menos habitual, dedicar parte de su actividad a este tipo de desarrollos personalizados, los cuales consiguen que una funcionalidad en el pasado fuera del alcance de la mayoría sea hoy una realidad accesible. El proyecto ahonda en el uso de diferentes metodologías y tecnologías web, e involucra no sólo el análisis y construcción de una herramienta que cumpla las necesidades que un usuario final pudiera plantear sino que la amplía dotándola de capacidad de integración con otras ya existentes. De esta forma, ampliando la funcionalidad básica mediante el uso integrado de interfaces de redes sociales, se obtiene un producto más rico y con más posibilidades que las que se conseguirían con una web estática y aislada, sin relación con otros sistemas.

Por tanto este proyecto cumple los objetivos que se le pueden encomendar a un trabajo de este tipo. Pretende mostrar dominio de conceptos abordados durante la carrera a la vez que explora algunos puntos algo más alejados de lo meramente académico, como puede ser la integración de un producto con redes ya existentes o el aprendizaje y uso de nuevas tecnologías. Intenta tener un enfoque orientado al mercado laboral ya que perfectamente podría haber sido un encargo realizado por cualquier particular, y el volumen de trabajo realizado cumple con los estándares habituales de un Proyecto Final de Carrera.

1.2. Objetivos

El objetivo de este proyecto es, partiendo de una base existente, modernizar y publicar una plataforma web de gestión de un equipo de baloncesto. En la web se deben poder consultar estadísticas de los partidos disputados por el equipo en diferentes formatos, así como servir de punto de encuentro digital para los componentes del mismo. Por supuesto, la intención es montar el sistema completo y que sea plenamente funcional, accesible para diferentes usuarios y estable. Por ello entra también en el proyecto la instalación y configuración de un servidor web, el establecimiento de un direccionamiento web y todo aspecto relacionado con una correcta usabilidad del sistema.
Se pretende combinar el sistema con la pre-existente y muy extendida red social Facebook, aplicando enlazado de cuentas para la identificación de usuarios y, en un futuro, autopublicaciones.
Para llevar a cabo este objetivo, se parte de un Framework PHP, se le dota de la funcionalidad requerida, se actualiza el código para ajustarse a las versiones modernas y se plantea una manera de moderna de distribuir, publicar y ejecutar la aplicación mediante contenedores Docker.

El objetivo es que la aplicación web conste de diferentes módulos y tenga una estructura sencilla y usable. Debe proporcionar al usuario las herramientas que necesite para la gestión del equipo y presentar la información de una manera cómoda.
Al final, se quiere demostrar que partiendo de una base común se puede crear un producto ajustado a las exigencias de un cliente aportando valor añadido y excediendo los límites iniciales de dicha base.
A nivel académico, la intención es demostrar dominio de las herramientas de desarrollo, conocimiento de los conceptos necesarios y el uso apropiado de las diferentes tecnologías que integran el sistema y que son detalladas en los capítulos correspondientes, así como capacidad de adaptación a actualizaciones, nuevos requerimientos, etc.
En particular, se plantea el uso de un framework que ofrece un esqueleto que actúa como punto de partida, a partir del cual se introducen los módulos necesarios. El framework dota de maneras ágiles de conectar con base de datos, una estructura básica y objetos que implementan diferentes funciones imprescindibles.

Al plantear como objetivo aplicar un enlazado de cuentas se pretende estudiar la mejor solución para que un mismo usuario de diferentes sistemas pueda ser identificado de manera segura en todos ellos de una forma más cómoda, utilizando un solo juego de credenciales para el acceso a varios de

ellos.

Finalmente, el uso de Docker permite investigar maneras rápidas y seguras de ejecutar el código en diferentes entornos utilizando el mínimo de recursos imprescindible, como se verá en su capítulo correspondiente.

1.3. Contenido de la memoria

El presente documento se divide en varios capítulos.

- Estado del arte: en este capítulo se repasa los orígenes y estado actual de los diferentes aspectos del desarrollo web, lenguajes de scripting, bases de datos, gestión de sistemas, etc.
- Desarrollo original de la aplicación: se detalla el análisis de requisitos, los diseños del modelo de datos y de la aplicación, el desarrollo del código y la integración con Facebook.
- Migración del código: en este capítulo se habla de las tareas que fueron necesarias para actualizar el código original a la versión moderna del lenguaje utilizado.
- Dockerización: en él se documenta el proceso seguido para preparar la aplicación para su uso con Docker.
- Securización del servidor: aquí se habla de la configuración del servidor para que la navegación por la página sea segura y los datos viajen cifrados.
- Pruebas: capítulo donde se detallan las pruebas realizadas para dar por buena la aplicación.
- Historia del proyecto: en él se hace un repaso por todas las fases que ha tenido el proyecto a lo largo del tiempo, las dificultades encontradas durante el mismo y las posibles soluciones valoradas y adoptadas.
- Conclusiones y trabajos futuros: capítulo donde se analiza el trabajo realizado y se plantean posibles líneas de ampliación y evolución para la plataforma.
- Manual de instalación: instrucciones para instalar la aplicación.
- Manual de usuario: espacio dedicado a especificar cómo se espera que los usuarios naveguen por la aplicación.

- Bibliografía: referencias bibliográficas útiles relacionadas con el proyecto.

Capítulo 2

Estado del arte

Actualmente hay un importante sector laboral orientado al desarrollo web, existiendo numerosos métodos de creación de páginas HTML y una amplia variedad de tecnologías y lenguajes adscritos a este ámbito.

2.1. Los orígenes del desarrollo web: inicio

Comenzando con algo de visión histórica, la primera página web HTML fue escrita en el año 1991 por el ingeniero Tim Berners-Lee y alojada en el primer servidor web del mundo situado en el CERN [1]. Se trataba de un documento de menos de 80 líneas de código el cual servía de puerta de entrada al entorno web, por medio de un serie de enlaces que contenían información sobre el lenguaje, el proyecto llevado a cabo, etc.

Estaba desarrollada, por tanto, con una estructura clásica HTML incluyendo los tags básicos: `<head>`, `<body>`, `<h1>` y demás. Ese fue el inicio del camino, y las primeras páginas web se ajustaban a un modelo monolítico en el cual un único fichero contenía todo el código que necesitaba un navegador para mostrar la página. Se realizaba una única petición GET al servidor el cual enviaba el documento entero para ser mostrado. Contenidos básicos como por ejemplo imágenes jpg en bruto eran referenciados directamente con elementos `<img>` y descargados en cada navegación, y los diseños se reducían a usar los atributos que proporcionaba el lenguaje HTML (colour, background, font, etc.). Las únicas animaciones que podían aplicarse eran a través del elemento `<marquee>`, no excesivamente manejable ni demasiado atractivo visualmente. El desarrollo se reducía, por tanto, a la redacción de documentos .html aprovechando las características ofrecidas por el propio lenguaje y su enlazado mediante referencias. Estas páginas web estáticas

junto con las conexiones tan lentas de la época, en la cual imperaban los módems de 56 kbps, hacían de la navegación web una experiencia muy alejada de lo que podemos vivir hoy en día.

2.2. Mejoras visuales: la introducción de Javascript y Flash

Un salto cualitativo llegó cuatro años después, en 1995, con la llegada del lenguaje JavaScript. El origen de esta solución se encuentra en la tecnología EaseScript, implementada por Netscape mediante su lenguaje LiveScript. Al menos ese era el nombre original, aunque finalmente decidieron sustituirlo dado que la palabra *Java* estaba de moda esos años al ser el nuevo paradigma que se imponía en el mundo de la programación. Pocos meses después de la introducción de esta novedad, Microsoft sacó su propia versión (JScript) y para intentar coordinar una evolución común el organismo ECMA (European Computer Manufacturers Association) decidió estandarizar el lenguaje en el año 1997, dando luz al estándar ECMA-262 [10]. En él se definía el ECMAScript, versión estándar del original de Netscape. Posteriormente la ISO adoptó el estándar, de manera que a nivel global se definieron las condiciones y especificaciones mediante el ISO/IEC-16262 [11]. La realidad es que el intento no resultó todo lo exitoso que se pretendía, puesto que el Internet Explorer de Microsoft (a la postre el navegador dominante a nivel mundial gracias a su incrustación en las diferentes versiones de Microsoft Windows, sistema operativo más utilizado) no implementó el estándar plenamente. Debido a ello, los desarrolladores aún a día de hoy deben modificar sus scripts de tal manera que tengan en cuenta si están siendo ejecutados por Internet Explorer o por otros navegadores que sí siguen el estándar.

Por lo tanto, lo que comúnmente se conoce como JavaScript es la implementación del estándar ISO/IEC-16262, y se trata de un lenguaje orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico; cuyo uso empezó a extenderse por su capacidad de ejecutarse en los navegadores web, en el lado cliente, ampliando las posibilidades de las interfaces de usuario y mejorando el aspecto visual de las anteriormente estáticas páginas web. Algunas de las ventajas que ofrece son la posibilidad de realizar cálculos directamente en el navegador del cliente, lo cual es útil, por ejemplo, para evitar el envío de datos inútiles o erróneos al servidor y la posterior recarga

de la página (recordemos las bajísimas velocidades de la época); o para dotar de mejoras visuales a las webs mediante animaciones, efectos visuales, barras de progresos, pestañas, etc.

Prácticamente al mismo tiempo, en el año 1996, surgió la tecnología conocida comúnmente como Flash. La empresa Macromedia, posteriormente Adobe Systems, creó un entorno de creación de gráficos que mejoraba lo existente, el Adobe Flash. Con él se podían crear archivos SWF que posteriormente podían ser reproducidos con el Adobe Flash Player. La ventaja que ofrecían era la distribución del player en forma de plugin, un objeto de ActiveX que otorgaba la posibilidad de reproducir los archivos en un navegador web. Estos archivos se basan en generar imágenes mediante gráficos vectoriales, lo cual reduce considerablemente el ancho de banda necesario para su transmisión. Eso, unido a un entorno con soporte para un lenguaje de programación llamado ActionScript (basado en el estándar ECMAScript) dio lugar a una amplísima implantación de la tecnología en el mercado de la navegación web. Su punto álgido llegó con la inclusión del plugin de Flash Player por defecto en las distribuciones de Windows XP y en todos los navegadores web comerciales, llegando a un 92 % de instalaciones entre los usuarios.

Estas dos mejoras supusieron un tremendo cambio en el ámbito del desarrollo web puesto que introducen las bases para abordar muchas más posibilidades de las que en principio se podían pretender con el HTML puro original. Por ejemplo la reproducción de archivos multimedia (vídeos, música, galerías de imágenes) o juegos directamente on-line vieron incrementado su impacto y su uso. A día de hoy, Flash está abandonado y los principales sistemas están dejando de ofrecer compatibilidad con él, pero es innegable su aportación al mundo web. No ha desaparecido del todo de Internet, aún es posible encontrar páginas poco actualizadas que se hicieron en su día utilizando esta tecnología. Eso nos da una idea de lo importante que fue su aparición para acercar las funcionalidades web al gran público.

2.3. El nacimiento de los lenguajes de scripting y CSS

El siguiente avance y pieza clave en este proyecto fue la aparición de tecnologías y lenguajes de ejecución en lado servidor como ASP, JSP y PHP. Aparecieron ante los cambios que iban surgiendo en el campo web, para ofrecer alternativas. Todos son lenguajes que generan HTML mediante un código que se ejecuta en el servidor antes de enviar el documento al cliente

que lo ha solicitado. Esto permite generar los documentos dinámicamente respondiendo a datos enviados mediante formularios, parámetros de la llamada, etc. El caso de JSP es el camino que tomó Sun con su lenguaje Java, ASP fue la implementación de Microsoft que inició su andadura en 1998 y PHP (Hypertext Preprocessor) se trata de una alternativa de código abierto. Estas opciones están más que vigentes hoy día, un altísimo porcentaje de las webs actuales se escriben utilizando alguno de estos lenguajes.

PHP es la opción elegida para este proyecto por su gratuidad y su facilidad de uso. Sólo se necesita instalar un motor PHP en el equipo que funcione como servidor, y al pedir por http un fichero primero se ejecuta el código que contenga, generando un HTML que es enviado al cliente final para ser mostrado. Su primera versión data de 1995 pero no fue hasta 1997, con PHP3, que se empezó a popularizar, gracias a la inclusión de plugins de bases de datos como MySQL y PosgreSQL; y Apache. Más tarde aparecerían PHP4 (2000) con soporte de multisesión, PHP5 (en 2004, con sus diferentes versiones) y finalmente el actual PHP7 en diciembre de 2015 (la última versión, 7.1.9, fue liberada a finales de agosto de 2017) [18]. Ofrece multitud de plugins (aparte de los que facilitan el acceso a base de datos, usado en este proyecto) y se puede combinar con tecnologías adyacentes. Viene al caso citar AJAX (Asynchronous JavaScript And XML) que permite recargar una web modularmente, es decir, a trozos. Esto dota de una agilidad mayor a la web puesto que permite actualizaciones más rápidas, menor tiempo de carga y una amplia variedad de posibilidades. Apareció alrededor de 2005 y utiliza documentos XML asíncronos para el intercambio de información entre en el servidor y el cliente. [22]

Caminos similares fueron tomados con lenguajes como Ruby o Ruby on Rails, lenguajes de scripting que también se han utilizado para desarrollos web, pero quedan fuera del ámbito de este proyecto.

Otro aspecto fundamental de la evolución del sector fue la introducción de CSS (Cascading Style Sheets).La primera recomendación oficial a este respecto, CSS1, vió la luz a final de 1996, y la segunda en mayo del 98. A partir de entonces ha tenido una gran distribución y aceptación. Su valor principal es separar vista de contenido en una página web. Esto facilita el diseño y el mantenimiento, porque permite modificación de estilos, fuentes, colores etc. de una manera más limpia. El lenguaje consiste en asignar una serie de valores a diferentes parámetros del diseño de una web en ficheros separados del código HTML. Por ejemplo colores, fuentes, tamaños,

márgenes; todos pueden ser controlados de esta manera. Ello permite un diseño basado en capas (divs) en lugar de una maquetación mediante tablas, la norma establecida hasta entonces.

2.4. Popularización del desarrollo: gestores de contenidos y plantillas web

Posteriormente comenzaron a aparecer nuevos paradigmas en el ámbito de la creación web. Aunque no formen parte de este proyecto, formaron parte importante de la evolución del desarrollo web. Por citar un caso que ejemplifica claramente esta situación, hablaremos brevemente de la plataforma Joomla. Joomla se trata de un CMS (Content Management System) o gestor de contenidos que apareció en 2005 [12]. Facilita la gestión de sitios web sin necesidad de una alta formación técnica, puesto que integra en un panel las herramientas básicas y avanzadas que un usuario pueda necesitar. Está escrito en PHP y utiliza un servidor Apache y una base de datos, la más común MySQL. Existe una amplia comunidad de desarrolladores que trabajan en Joomla, de tal forma que el soporte es muy accesible y rápido, y se presentan módulos y actualizaciones muy a menudo. Utilizando las herramientas que proporciona se puede reutilizar código ya escrito y probado para generar webs de muy diferentes características, desde foros a tiendas online, pasando por blogs, agregadores de noticias, etc. Existen multitud de plantillas, extensiones y plugins que pueden ser agregados directamente a través de un panel de control para crear, mantener y actualizar cada sitio. Esta flexibilidad y potencia ha dotado al CMS de una gran popularidad. Por ejemplo empresas tan grandes como Ikea o Ebay tienen sus sitios webs auspiciados por Joomla [9], así como decenas de miles de pequeños desarrolladores y otras empresas de menor tamaño. Por lo tanto, sus ventajas son la reunión de todos los elementos necesarios bajo un mismo ámbito y un potente soporte proporcionado por la comunidad. Como desventajas tenemos la falta de un control absoluto sobre el código y problemas derivados de versiones, componentes y demás. Al ser código preescrito, no es fácil personalizarlo ni modificarlo, y no se controla exactamente. A nivel de diseño, ofrece numerosas vistas preescritas, pero para ir más allá de cambiar colores o emblemas se hace algo complicado el variar un volumen alto de código autogenerado. Además no es tan directo integrar nuevas funcionalidades porque pueden provocar incompatibilidades, problemas de versiones, etc. Resumiendo, para un uso básico y rápido es muy útil y eficiente utilizar este tipo de software agregador, sin embargo cuando

se desea un producto más complejo y adaptado a características algo más específicas suele ser mejor un desarrollo a más bajo nivel, en el cual se tenga un control más completo sobre los diferentes aspectos del sistema.

La aparición de estos gestores ha dado pie a soluciones aún más simplificadas como las que ofrecen empresas tales como 1and1 y otras similares. Basándose en estas tecnologías ofrecen webs-plantillas con pequeñas opciones de personalización (de nuevo logos, colores y demás) con la ventaja de ser un sistema muy rápido y directo, en pocos minutos puedes tener una web moderadamente simple. Esta aproximación no es desarrollo web como tal, más bien parte de una base común y ofrece una funcionalidad limitada a personas o entidades que no tienen necesidad de productos más complejos. Ofrece la posibilidad de cambiar detalles como títulos, formas y colores y permite generar webs de tipo informativo rápidamente. Opciones simples como subida de fotos, inserción de textos y comentarios, etc. también están soportadas, pero ir a personalizaciones más complejas queda fuera sus posibilidades.

2.5. La evolución hacia los dispositivos móviles

El siguiente gran salto en el ámbito de los desarrollos web llegó con la aparición y posterior proliferación de dispositivos con conexión a internet y capacidad para visualizar páginas web: Smartphones y Tablets. En un principio el desarrollo web estaba orientado únicamente a ordenadores, puesto que eran las únicas máquinas que podían dar acceso a Internet, pero en los últimos años los Smartphones y Tablets han ido ganando cuota de mercado hasta llegar a cifras de tráfico de alrededor de un 55 % del volumen total [13]. Los navegadores web de estos *gadgets* están adaptados a sus diferentes características, como pueden ser pantallas más pequeñas (de 4 a 10 pulgadas frente a las 17 en adelante que presentan típicamente los monitores) y conexiones con anchos de banda más limitados. Esto es debido a que, por lo general, están orientados a conexiones a través de redes móviles como UMTS o LTE, que dependiendo de la estabilidad de la conexión y la cobertura pueden ofrecer velocidades más bajas que conexiones domésticas a través de ADSL o fibra óptica (ambas componen la práctica totalidad de las conexiones en hogares y empresas). Además estas conexiones suelen presentar volúmenes de descarga de datos limitados a ciertos valores mensuales. Estas especiales características han forzado a

los desarrolladores a adaptar sus diseños para optimizar las experiencias de los usuarios. El salto ha sido facilitado por los lenguajes de ejecución en el servidor ya comentados, que permiten generar un código u otro en función del navegador a quien vayan dirigidos; y por las facilidades que ofrece CSS, que permite cargar una vista u otra también en función del navegador que lo solicite. Así, versiones más simplificadas pueden ir dirigidas a los Smartphones, limitados por consumo de datos y pantalla; otras con un diseño diferente para Tablets, orientadas en general de forma apaisada para la navegación; y versiones clásicas completas para ser expuestas en ordenadores personales o portátiles. A día de hoy empiezan a extenderse otras opciones, los llamados wearables, como pueden ser SmartWatches o los primeros prototipos de Google Glasses. Se trata de aparatos que hasta ahora eran complementos en el ámbito de la moda que comienzan a incorporar conexión a internet, por lo que no es en absoluto descartable que sea necesario un nuevo ajuste de los formatos para adaptarlos a las nuevas pantallas. También empiezan a crecer las conexiones M2M (Machine to machine), que componen el llamado internet de las cosas, pero este tipo de tráfico va más orientado a intercambio de datos, telemetría y controles que al consumo de contenidos web de un usuario normal, por lo que no afecta a lo aquí expuesto.

2.6. Nuevo estándar web

Unido a estos cambios llegó la siguiente evolución de los estándares, siendo la más importante y ya ampliamente implantada el HTML5. Se trata de una evolución del HTML que fue oficialmente presentada por el W3C (*World Wide Web Consortium*, comunidad internacional dedicada a elaborar estándares y recomendaciones para las tecnologías web) en diciembre de 2013, y que fue estandarizada en octubre del año 2014. Su propósito es reunir bajo un lenguaje común las soluciones a las distintas debilidades que sufría el HTML original y que se han ido solventando a lo largo de los años con diferentes tecnologías adicionales, como se ha expuesto en páginas anteriores. Así, pretende integrar las posibilidades ofrecidas por Flash y otros plugins de manera nativa, sin dependencia de terceros; ofrecer soporte para integrar contenidos multimedia directamente, así como utilizar los códecs necesarios para reproducirlos; y simplificar la estructura de los documentos, haciéndolos menos complejos y más manejables, además de más optimizables. Además, intenta potenciar las posibilidades de ejecución de scripts (JavaScript, por ejemplo) mediante nuevas posibilidades tanto de almacenamiento como de ejecución en paralelo, dotando así de más potencial a los desarrollos. Otro

avance es la mayor integración con una nueva versión de CSS (CSS3), con nuevas posibilidades de diseño y adaptación a diferentes dispositivos, así como la introducción de más efectos visuales nativos mediante canvas. Ello facilitará adaptaciones a diferentes formatos y otorgará flexibilidad para el futuro. Por otro lado, aporta funcionalidades que antes no existían, como geolocalización directa; y añade nuevas etiquetas orientadas a nuevas formas de posicionamiento y búsqueda, la llamada web semántica, a través de metadatos.

Esta web semántica es otro de los avances futuros más importantes, y aunque queda lejos de lo planteado en este proyecto, merece la pena un pequeño comentario al respecto. Se trata de un nuevo paradigma en la forma de entender las relaciones entre documentos, de tal forma que cualquier documento accesible a través de la web contenga una serie de información que permita que una máquina la interprete y pueda así relacionar el documento con otros similares o relacionados, de manera lógica. En teoría su implantación supone una forma totalmente distinta de enfrentarse a búsquedas, integraciones, ventas, etc. Dado que estos conceptos quedan lejos del propósito de este documento, no se abordará en más profundidad.

2.7. Hosting y DNS

Finalizado ya el repaso a la situación histórica y actual del desarrollo web como tal, pasamos a comentar otra de las bases de este PFC. Se trata del hosting y el DNS. El hosting es el sistema por el cual una página es alojada en un servidor, ya sea este dedicado o virtual. Así, para que una página pueda ser solicitada por un cliente, algún servidor debe servirla. Existen varias opciones para el alojamiento, hay empresas dedicadas a ofrecer el servicio o se puede optar por realizar un mantenimiento propio en máquinas privadas o personales. La medida de calidad de un hosting se basa en el ancho de banda que ofrece, la disponibilidad y la capacidad de hardware. El ancho de banda debe preverse de tal manera que el servicio pueda ser prestado al número de clientes estimado dentro de márgenes de respuesta temporal establecidos. Igualmente, aunque una disponibilidad del 100 % es utópica, el servicio debe estar disponible (o *levantado*) el máximo tiempo posible. Empresas privadas como por ejemplo OVH ofrecen 99 % del tiempo de servicio. Respecto al hardware, debe diseñarse en función de las necesidades del servicio a ofrecer: discos duros, memoria RAM, procesadores, etc. son factores que pueden ser escogidos para adaptarse a las necesidades del producto. Un paso más allá del hardware dedicado en el hosting supuso

la computación en *cloud* o en la nube. A día de hoy, empresas como Google o Amazon ofrecen servicios de virtualización avanzada en la nube. Desde una sola cuenta se pueden gestionar multitud de equipos virtuales que, con la infraestructura adecuada, pueden crecer o decrecer en función de las necesidades del tráfico. Además los datos están fuertemente replicados para dotar de robustez a los sistemas, haciendo copias de seguridad distribuidas que pueden ser gestionadas manual o automáticamente.
Otro aspecto importante para el acceso a la web es el uso correcto del protocolo DNS (Domain Name System), que básicamente transforma direcciones legibles para las personas en direcciones IP para las máquinas. No se detallará el protocolo en sí dado que forma parte del contenido de la carrera, pero su uso es necesario para que una web llegue a ser accesible a sus destinatarios. Para ello, la web es accedida a través de un dominio (por ejemplo google.com). El navegador o el programa utilizado utiliza DNS para obtener la dirección IP del servidor. Los dominios pueden comprarse o alquilarse. También existen dominios gratuitos como por ejemplo los .tk. Lo habitual es que el propio sistema que otorgue el dominio ofrezca también una herramienta para redirigirlo a la dirección IP donde se vaya a realizar el hosting. En la actualidad el mercado de los dominios se ha ido saturando, no es suficiente con los tracionales .org, .com, .net y demás, por lo que se liberaron diferentes terminaciones como .tv, .fm y otros tantos. Un organismo en cada estado puede conceder los que indiquen su relación con el país, como pueden ser los casos de .es en España o .it en Italia, entre una gran variedad de ejemplos.

2.8. Bases de datos

Otro factor a tener en cuenta en el desarrollo del proyecto es el referido a la administración de datos, en este caso mediante bases de datos. Una base de datos consiste en una serie de información recogida de una manera estructurada común. En el ámbito informático, hablamos de una estructura que permite tener guardada una serie de datos en un formato particular, para poder ser mantenidos, consultados y ampliados con facilidad. Puede ir desde una lista de títulos de canciones guardados en un fichero de texto plano hasta un modelo completo de todos los datos que pueda manejar cualquier tipo de empresa guardados en un sistema Oracle, pasando por archivos de celdas u hojas de cálculo y pequeñas bases de datos definidas mediante Microsoft Access o similares. Una base de datos se caracteriza por el modelo de datos que implementa. Existen una multitud de modelos, cada uno con sus particularidades.Así, según la lógica que sigan, existen modelos

relacionales, de objetos, en estrella, jerárquicos, asociativos, semánticos, documentales, etc. Las bases de datos más extendidas en la actualidad son del tipo SQL. Son las siglas de Structured Query Language, un lenguaje que permite gestionar bases de datos con modelos lógicos relacionales, esto es, que establecen relaciones directas entre las diferentes entidades que componen el modelo. Las relaciones pueden ser uno a uno, uno a varios o varios a uno. Con SQL se pueden ejecutar una gran variedad de acciones sobre las bases de datos, desde recuperar datos que cumplan ciertas condiciones a insertar o modificar los ya existentes. Este tipo de bases de datos se estructuran mediante tablas que contienen registros o filas con los valores que se especifiquen en las columnas. Así, una tabla hace las veces de un contenedor de conceptos unitarios, y en ella se almacenan las instancias existentes. SQL se divide en dos grandes bloques, el lenguaje de definición de datos o LDD y el de manipulación de datos, LMD. Con LDD se maneja la estructura de la base de datos mediante las acciones para crear, modificar y eliminar tablas, mientras que LMD da las opciones para insertar, modificar o eliminar datos de esas tablas, así como de recuperar la información que contienen mediante consultar parametrizables. El uso de SQL permite asegurar la integridad referencial de la base de datos, es decir, que no existan incongruencias entre datos existentes o referencias a datos equivocadas; controla las acciones a nivel transaccional (es decir, ante errores se recupera el estado anterior a la ejecución del comando fallido mediante *roll-back*) y ofrece una capa de seguridad para limitar el acceso a los recursos mediante niveles de permisos. La variedad de sistemas de gestión de bases de datos SQL es muy alta: DB2, Firebird, Oracle, MySQl, PostgreSQL, SQLite y un largo etcétera. Cada uno es ligeramente diferente, utilizan motores distintos, las especificaciones varían y también presentan variaciones en la sintaxis que utilizan. Sin embargo, todas tienen como base común SQL. A parte de las bases de datos de tipo relacional SQL, se está extendiendo con fuerza el uso de otras bases de datos noSQL como MongoDB, basada en un modelo documental. En ellas, los datos no se estructuran en tablas que contienen registros, sino en documentos individuales que se relacionan entre sí. Es un paradigma distinto que ofrece ventajas a la estructura clásica ante nuevos escenarios (por ejemplo, para datos geográficos) en cuanto a rendimiento y velocidad. Otro ejemplo podría ser GSQL, un lenguaje que permite definir y utilizar estructuras de bases de datos basadas en grafos en lugar de tablas. Esto lleva las relaciones entre elementos a otro nivel, permitiendo utilizar datos organizados de una manera mucho más rápida que con el modelo tradicional relacional.

2.9. Redes sociales

Otro aspecto vital en el desarrollo de este trabajo es el uso de lo que se conoce como redes sociales. Las redes sociales son uno de los usos de las web que más han cambiado la sociedad en la que vivimos, junto con el acceso instantáneo a la información. En realidad, las redes sociales han existido desde el inicio de la Humanidad, puesto que las personas somos seres sociales. El círculo de personas con el que cada uno se relaciona constituye su propia red social. El contacto que mantiene con esas personas, la cercanía con ellas, todo lo que se comparte está limitado en la vida real por la distancia y el tiempo de cada uno. Perder contacto con viejos amigos o no mantener comunicaciones fluidas por falta de medios ha sido siempre habitual. La llegada de las redes sociales a internet supuso una demolición de estas barreras, ya que simplemente otorgan esos medios de los que antes se carecía. Así, proporcionan plataformas web a través de las cuales cada persona puede comunicarse con quien considere que forma parte de su vida.

La red social por excelencia hoy día es Facebook, lanzada el 4 de febrero de 2004. Su idea inicial era simple: proporcionar un punto de acceso común a través del cual poder dejar mensajes a tus amigos y compartir contenidos multimedia con ellos. Antes de Facebook ya existían otras herramientas web que otorgaban estas posibilidades, como por ejemplo la plataforma MySpace o el Messenger de Microsoft.

Facebook evolucionó esas estructuras existentes dotando de un espacio personal (o muro en su terminología) en la cual poder publicar textos, fotografías y vídeos, incluyendo un capa de relaciones sociales en la que estos espacios se relacionaban a nivel lógico con los espacios de las personas que conforman el círculo social de cada uno. También integraría un servicio de mensajería instantánea que acabó desbancando al Messenger. Facebook se benefició de estar en el momento justo en el lugar apropiado, ya que la universalización del acceso a Internet favoreció en gran medida su extensión. Como red social, su principal atractivo es su número de usuarios, ya que cuantas más personas lo utilicen mayores opciones para que cada uno tenga contacto con sus conocidos (para ello, todos deben ser usuarios). Cada nuevo usuario suponía un efecto llamada para otros que derivó en conseguir para Facebook la posición dominante de la que ha gozado en los últimos tiempos, con más de mil millones de cuentas en 2013.
Aunque la principal funcionalidad de Facebook fue en un principio la comunicación entre personas mediante mensajes y fotografías, tardó poco en crecer y evolucionar. En 2008 lanzó un API llamado GraphApi junto con dos SKD's para interactuar con él (uno Javascript y otro PHP). El objetivo

era que desarrolladores interesados pudieran crear aplicaciones dentro de la propia plataforma, y así Facebook amplió sus posibilidades mediante juegos, herramientas para sorteos y promociones, tiendas y muchas otras.

El API es de tipo RESTful, por lo que no tiene sesión y acepta las funciones básicas del protocolo HTTP: GET, POST, DELETE y PUT. La funcionalidad del API se basa en llamadas GET a una URL especificando la acción a realizar, en general pedir información o subirla. Se puede buscar contenido, listar usuarios, obtener información de amigos de usuarios, publicar fotos, recuperarlas...las posibilidades son enormes. Su seguridad se basa en OAuth, funciona mediante el uso de tokens con expiración para identificar a quien realiza la llamada, y en función de los permisos asignados responder o no. Para este proyecto, este API es imprescindible, ya que es el que se utiliza para validar la identidad del usuario que intenta acceder a la plataforma.

Facebook no es sólo valioso para sus usuarios por las posibilidades que les ofrece, también tiene valor el gran volumen de información del que dispone. Toda información personal es valiosa para grandes empresas a nivel de marketing, publicidad, estudios de mercado, etc. El hecho de tener acceso a tanta información (gustos, hábitos, relaciones, horarios...las posibilidades son enormes) es de gran valor para entidades externas. Así, la privacidad se convierte en un importantísimo caballo de batalla para este tipo de servicios. En todo caso, a partir del éxito alcanzado y de su inmenso número de usuarios, y aprovechando la posibilidad de incluir aplicaciones personalizadas dentro de la propia plataforma, comenzó a extenderse la posibilidad de identificarse en diferentes sitios web con las credenciales de Facebook. De esta forma, Facebook confirma a otras empresas o servicios que un usuario es quien dice ser, puesto que tiene muchos datos recopilados suyos. Estas facilidades derivan en los enlazados de cuentas, o lo que es lo mismo, la posibilidad de relacionar cuentas de un mismo usuario para diferentes servicios en un único ámbito.

2.10. Autenticación mediante OAuth

OAuth (*Open Authorization*) es un protocolo abierto que permite la autorización segura de usuarios de aplicaciones. Su objetivo es proporcionar una manera segura y estandarizada de compartir información de usuarios entre dos servicios, de forma que un usuario de uno de ellos pueda utilizar el otro sin tener que compartir su identidad completa. Utilizando este protocolo

se obtienen *tokens* que permiten el uso de las aplicaciones. Así, se consigue otorgar acceso al API de un servicio desde otra aplicación de un tercero.

OAuth nació en octubre de 2007, y su versión 2.0 vio la luz cinco años después, en octubre de 2012. Grandes empresas como Google, Facebook y Github hacen uso de OAuth 2.0 para autenticar a sus usuarios y permitir que otros servicios (como por ejemplo la aplicación realizada en este PFC) puedan identificar a los usuarios de manera segura.

En OAuth se definen cuatro roles [17]:

- *Resource owner*: el usuario, la persona que da acceso a su información.
- *Resource server*: el API, el conjunto de funciones a las que se quiere tener acceso mediante el uso de token OAuth.
- *Servidor de autorización*: puede ser el mismo servidor que proporciona el API o no. Se trata del elemento con el que el usuario interactúa cuando una aplicación solicitia utilizar la información de su cuenta. Se encarga de generar los tokens de acceso para el cliente una vez el usuario ha dado su permiso.
- *Cliente*: la aplicación, es quien hace uso del API una vez el usuario ha dado su permiso

Existen dos tipos de clientes definidos en el protocolo, confidenciales y públicos. La diferencia entre ellos es su capacidad para mantener la confidencialidad del `client_secret` (los primeros pueden hacerlo, por ejemplo una aplicación web; los segundos no, por ejemplo aplicaciones Javascript que tienen el código en el propio navegador o aplicaciones móviles).

El protocolo definde tres conceptos básicos para poder hacer la identificación:

- Access Token: cadena de caracteres que el cliente envía al API para realizar acciones de manera autentificada. Se obtiene una vez el usuario ha dado su permiso al servidor de autorización, y tiene un ámbito y una duración determinados.
- Refresh Token: cadena de caracteres para obtener un nuevo access token una vez ha caducado el original. Es opcional, por lo que depende de la implementación del protocolo en particular.
- Authentication Code: se trata de un código intermedio que el servidor le envía al cliente para que pueda ser intercambiado por un access token.

Para poder utilizar OAuth, es necesario que exista una aplicación registrada en el sistema que provee el API. Lo más habitual es que el desarrollador registre su aplicación utilizando las herramientas que suele proveer el sistema que haya implementado el protocolo. En la aplicación se definen datos como su nombre, dirección web, etc. Este proceso desemboca en la obtención de un `client_id` y un `client_secret` que serán utilizados durante el proceso de autenticación. En la aplicación también se registran una o varias URLs de redirección, que es donde el servidor de autorización redirigirá al usuario una vez ha dado su permiso para que el cliente acceda a sus datos. El esquema de estas URLs debe ser HTTPS para evitar que la información sea interceptada, lo cual dota al sistema de robustez. Además, al estar registradas en la aplicación, se evita que el usuario pueda acabar redirigido a sitios maliciosos.

En resumen, el cliente lanza una petición al servidor de autorización, el cual, mediante aceptación del usuario, le devuelve un token de acceso. Con él, el cliente puede lanzar peticiones seguras al API para realizar las acciones que le hayan sido permitidas.

En este proyecto se hace uso de las herramientas que Facebook pone a disposición de los desarrolladores para implementar un sistema de autenticación con OAuth. Mediante la utilización de su SDK y su interfaz se dan todos los pasos necesarios para autenticar a los usuarios.

2.11. Virtualización

Para completar el estado del arte, sería necesario escribir unas líneas sobre la gestión de sistemas, ya que son el pilar sobre el que se levantan las aplicaciones. El modelo clásico consistía en un servidor, una máquina física, sobre la que instalaba software específico para dar cabida a las aplicaciones que se quisieran ejecutar. El problema que ofrece este modelo es que es muy complicado de mantener y de ampliar. Cada máquina debía ser actualizada manualmente, su software configurado, etc. Esto daba lugar también a problemas de compatibilidad, diferentes funcionamientos de los programas, etc. Para tratar de paliar estos problemas comenzó a trabajarse con máquinas virtuales, utilizando programas como VMWare o Virtualbox. Gracias a ellos se consigue replicar cualquier sistema operativo configurado como se desee, igualando las condiciones entre entornos con mucha más facilidad. Una máquina virtual consiste en una réplica de una máquina física utilizando los recursos de otra. Ejecuta un sistema operativo completo, su sistema de ficheros, etc. sobre el hardware de su anfitrión. Al ser un sistema completo su tiempo de carga y su consumo de recursos

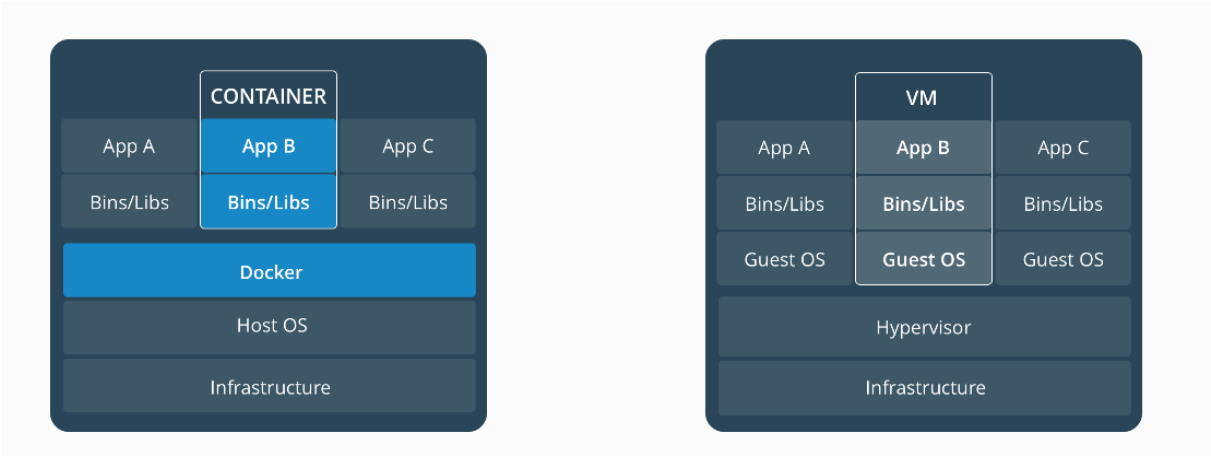

Figura 2.1: *Esquema de arquitectura Docker*

es elevado, pero dado el aumento de potencia del que se beneficiaron las máquinas supuso un salto importante, añadiendo facilidad a los desarrollos. Facilita también la actualización de sistemas, ya que instalando la imagen de la máquina resultante se puede tener replicada en varios lugares rápidamente.

La tecnología de contenedores o *containers* de Docker lleva esta idea un paso más allá. Según la propia definición que se encuentra en [4], un *container* es una manera de empaquetar software de tal manera que pueda ser ejecutado de manera independiente en un sistema operativo compartido. La idea es que para ejecutarlo no sea necesario un sistema completo, como en su modelo predecesor de máquinas virtuales. Sólo necesita las librerías correspondientes y la configuración, en cada caso, para que el programa funcione. La filosofía de Docker es que cada container ejecute un único hilo, aligerando de esta forma muchísimo los sistemas, consiguiendo que sean ligeros, independientes y eficientes. Además se consigue certificar que el software funcionará del mismo modo sin importar dónde esté desplegado: cualquier máquina con el software de Docker puede lanzar contenedores que ejecuten esas imágenes con sólo un comando, lo que permite manejar la gestión de los entornos de una manera mucho más moderna y rápida. La escalabilidad se multiplica, al poder ejecutar más o menos contenedores en función de la carga del sistema. Esto aplica tanto en horizontal como en vertical, al ser cada contenedor un único proceso: se pueden ejecutar más sólo del que haga falta, o crecer en conjunto todos.

Como se representa en la figura, un *container* es una abstracción en

la capa de aplicación que empaqueta el código y sus dependencias. De esta forma es posible ejecutar diferentes *containers* en la misma máquina, compartiendo el kernel del sistema operativo, estando cada uno de ellos aislado en su propio espacio. Es por ello que los containers ocupan mucha menos memoria que las máquinas virtuales, consumen menos recursos y son más rápidos. Como beneficio añadido arrancan casi instantáneamente, ya que no hay que ejecutar el sistema operativo para cada uno de ellos.

Docker vio la luz en 2013, distribuido como código libre. Su escalada ha sido espectacular, en menos de dos años se coló en el top20 de proyectos con más estrellas de la plataforma Github (un popular repositorio de código utilizado ampliamente por la comunidad de desarrolladores) gracias a las aportaciones de grandes corporaciones como Google, Red Hat, Cisco y Microsoft, entre otras. Docker ha sido descargado más de trece mil millones de veces, según [21].

Capítulo 3

Desarrollo original de la aplicación

El presente PFC se divide en dos grandes bloques fuertemente diferenciados: el desarrollo original de la aplicación y la evolución posterior de la misma, pasado el tiempo.

Este primer bloque consistió en la programación original de la funcionalidad de la aplicación. Para ello, fue necesario atacar tres necesidades:

- Definir un modelo de datos y elegir una tecnología que le diera soporte.
- Desarrollar el código que implementa las funcionalidades deseadas.
- Hacer la página visible para los usuarios.

3.1. Requisitos funcionales de la aplicación

En este punto detallo la funcionalidad que diseñé para la aplicación.

Hay dos tipos de usuarios, los usuarios normales (con rol usuario) y los administradores. La aplicación debe poder identificarlos y diferenciarlos, por lo que se requiere la capacidad de hacer login. Además de su rol, un usuario puede estar baneado o no, es decir, debe existir la funcionalidad para restringir el acceso.

Para entrar a la aplicación, un usuario debe introducir sus credenciales. Si no dispone de ellas, puedes darse de alta usando el botón de Registrar o entrar directamente con su cuenta de Facebook. Simplemente, si es la primera vez, se introduce el usuario en la ventana del navegador que se solicita, se aceptan los permisos de la aplicación y ya se puede navegar con normalidad. Una vez identificado correctamente, se puede navegar por

los diferentes módulos del sistema haciendo click sobre las opciones que se deseen consultar. Si el usuario está baneado, sus credenciales no proporcionan acceso, se queda en la pantalla inicial con un mensaje indicando el problema.
La información que la aplicación debe ofrecer sería una tabla resumen con los líderes estadísticos de cada categoría como inicio, y luego información específica. Se ofrece información por partido, por jugador, por categoría estadística y los datos del equipo a nivel general.

Otras funcionalidades son modificar los datos del propio usuario y salir de la aplicación.

Un usuario administrador entraría con sus credenciales y, de la misma forma, puede navegar por los módulos. Además de esto puede gestionar los datos del sistema, cada uno en su módulo correspondiente.

Por último, la aplicación debe ser capaz de almacenar y mostrar por separado, de manera ordenada, datos de diferentes temporadas. Por defecto, los datos de la resumen son los de la temporada por defecto, la cual se indica como variable global en el fichero de definiciones.

3.2. Diseño del modelo de datos

El primer paso fue elegir qué datos eran necesarios para ofrecer la información que se planteaba como objetivo. Al ser estadísticas de un equipo de baloncesto, es lógico organizar los datos en torno a jornadas. Esas jornadas se agruparían en temporadas, y estarían compuestas por las estadísticas individuales de cada jugador. De este hecho viene otro tipo de dato necesario: el propio jugador. De esta manera representamos la aportación estadística de cada jugador al equipo, en cada partido de cada temporada.

Los datos estadísticos que se recogen de cada jugador son básicamente sus tiros de dos anotados, sus tiros de tres anotados, sus tiros libres anotados, sus tiros libres intentados y sus faltas realizadas. Los tres primeros conforman los puntos totales anotados. El motivo de elegir estos datos y no otros también interesantes y clásicos como rebotes, asistencias o tapones es que estos datos no vienen recogidos en las actas oficiales de los partidos de liga local (el objetivo de esta plataforma es dar cabida a los números de un equipo amateur). No sería complicado añadirlos al modelo en caso de disponer de ellos.

Dado que un jugador puede cambiar su dorsal al cambiar de temporada, o que puede abandonar el equipo y otro ocupar su lugar (y por tanto poder

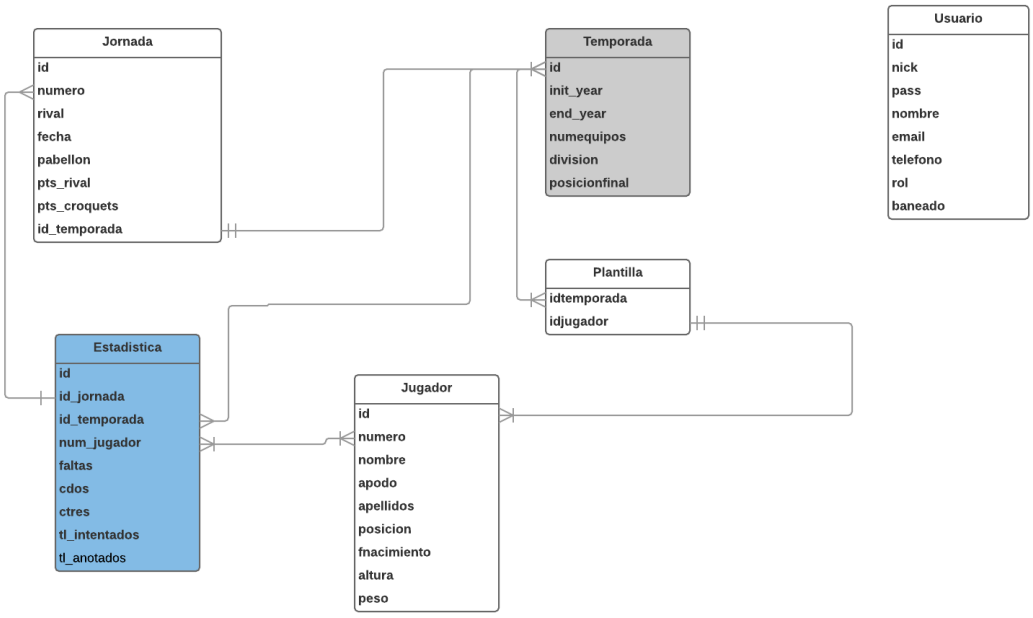

Figura 3.1: *Esquema Entidad-Relación*

elegir su dorsal), surge la necesidad de una tabla intermedia que relacione las temporadas con los jugadores que juegan en el equipo durante cada una de ellas: la tabla de plantillas.

Por último, es neceario un último tipo de dato que represente a los usuarios del sistema, para organizar el sistema de accesos y de roles. Dado el requisito de aplicar un modelo de roles para los usuarios para dotar de profundidad a la plataforma, es necesario definir dos roles básicos: administrador y usuario. Un usuario puede ver toda la información que muestra el sistema a través de su interfaz, y editar sus datos de usuario; pero no puede modificar ningún otro dato del sistema. El administrador, por el contrario, tiene permisos tanto de lectura como de escritura sobre los datos que maneja el sistema.

El modelo es claramente relacional, por lo que la elección de usar MySQL como tecnología de almacenamiento es apropiada. El principal motivo de elegir esta opción fue su gratuidad y sencillez de instalación y mantenimiento. Al no ser una aplicación de uso masivo de datos, el rendimiento no parecía un problema a priori, por lo que ir a soluciones más avanzadas parecía innecesario, MySQL cumple de sobra con los requerimientos del sistema.

El modelo de datos se compone por tanto de las siguientes tablas:

- estadisticas
- jornadas
- jugadores
- plantillas
- temporadas
- usuarios

Cada una de las tablas representa uno de los datos utilizados por el sistema. Los nombres son bastante explicativos, pero en temporadas se almacenan datos sobre las temporadas que se pueden consultar (año, división, etc.), en jugadores datos sobre cada uno de los jugadores que conforman la plantilla (dorsal, nombre, apodo, etc.), plantillas relaciona jugadores con temporadas ya que los dorsales y los jugadores pueden variar de una temporada a otra; en jornadas se guardan los datos sobre cada partido de una temporada (rival, resultado, fecha, etc.) y finalmente en estadisticas se guardan los números en sí que haya hecho cada jugador en cada jornada. La tabla usuarios contiene los datos de los usuarios que pueden entrar en el sistema, que no tienen por qué ser jugadores.

Una vez analizadas las necesidades, elaborado el diseño del modelo de datos y elegida la tecnología, instalé el servidor en mi ordenador local, creé la base de datos y las tablas correspondientes y procedí a ocuparme del código en sí.

3.3. Diseño de la aplicación

La aplicación está escrita de forma modular utilizando un esqueleto existente, un framework empleado en los desarrollos web realizados en la empresa Asiro Systems, para la cual trabajé entre los años 2011 y 2014 y para el cual colaboré en su creación. La aplicación sigue el patrón de diseño MVC, separando modelo, vista y controlador.

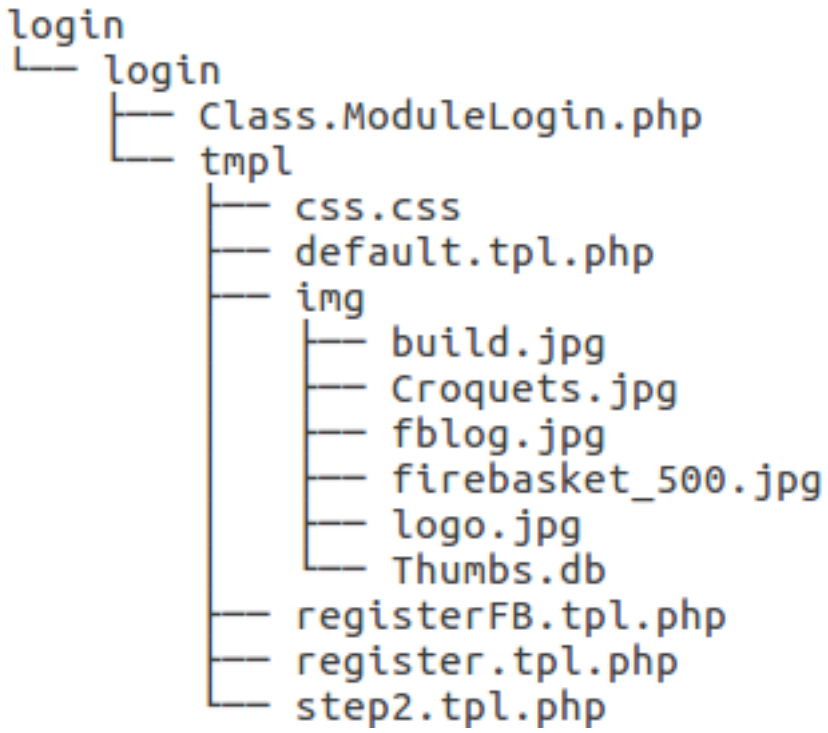

Figura 3.2: *Ejemplo de la estructura de un módulo del framework*

El modelo lo componen las clases de envoltorio para cada tipo de dato y las clases globales de utilidades, proporcionadas por el framework. También proporciona una clase `SModule` que es la clase padre de cada controlador: dentro de cada módulo existe una clase que actúa como controlador y unos archivos que determinan las vistas o *templates*. El controlador ejecuta toda la lógica, recupera los datos necesarios de la base de datos o realiza las acciones necesarias. Los archivos con las vistas utilizan esos datos para construir el código HTML que finalmente le será devuelto al usuario para que lo visualice en su navegador.
Por ejemplo, para el módulo de login usaríamos una estructura como la representada en la figura 3.2. En la figura se muestra la estructura de uno de los módulos del sistema, el cual contiene su clase controladora (`Class.ModuleLogin.php`) y la carpeta con las archivos necesarios para cada una de las vistas: cada uno de los archivos con extensión `.tpl.php`, el css particular del módulo y las imágenes que se usan en él.

La idea es que cada página esté compuesta por dos módulos básicos, que serían el menú principal y el contenido en sí. El primero sería el módulo `menu_principal` y el segundo el módulo `content`, que se podría subdividir

en su propio menú y el contenido que se le cargara según la navegación del usuario.

3.4. Explicación en detalle del código de la aplicación

La estructura funciona de tal manera que la página de acceso sea index.php. Al acceder a index.php, se crea una instancia de la clase `SApplication` y se invoca la función `start` de ese objeto. Dentro de esa función se cargan los plugins necesarios, a saber: un plugin de sesión, un plugin de base de datos, un plugin de login y un plugin de documento. En este contexto, un plugin es un objeto de una clase que contiene determinadas utilidades que la aplicación usa para generar la web que se devolverá al usuario.

El plugin de sesión consiste en un objeto de la clase `sSession`. Se utiliza para controlar la sesión del usuario de la aplicación, guardando sus credenciales y datos una vez ha accedido. El plugin de login se encarga de controlar los accesos. Compruebas las credenciales e indica si el usuario tiene acceso al sistema o no. El plugin de base de datos es un objeto que representa la conexión a la base de datos. Se utiliza en toda la aplicación para las operaciones de lectura y escritura de cada dato. El plugin de documento representa la página en sí que se le va a mostrar al usuario. Una vez creados todos, se invoca show del plugin `SDocument`. Esta función es la que en realidad genera el HTML que será devuelto. La clase que realmente contiene la información sobre lo que se va a mostrar es un módulo, que hereda de la clase `SModule`. Todos los módulos se almacenan dentro del árbol de directorios dentro de una carpeta con su mismo nombre, dentro de la cual está el archivo `nombreArchivo.php` que contiene el código php que se ejecuta para recopilar los datos que sean necesarios y ejecutar los cálculos en cada caso; y una carpeta tmpl. Dentro de esta carpeta están los css que pueda requerir el módulo y los archivos de template, que son los que contienen el diseño de la vista. Cada módulo puede tener varios pasos o vistas, y cada uno de ellas sería un template, descrito en su correspondiente `nombreArchivo.tpl.php`.
El plugin `SDocument` cargará un `SModule` u otro, en función de la invocación que se haga. Si se invoca a `index.php` sin ningún parámetro, cargará un módulo y una tarea por defecto. El parámetro task indica la tarea, paso, vista, que debe ejecutar el módulo. Su nombre debe coincidir con alguno de los archivos .tpl.php que se encuentren en el directorio tmpl del módulo

en cuestión. Por defecto, si el usuario no se ha logado (para obtener esta información se hace uso del módulo de login, `SLogin`), el módulo que se carga es el módulo de login (un módulo que presenta la página donde introducir las credenciales, no confundir con el plugin de login que controla los accesos en sí). Si se ha logado, se muestra el módulo por defecto Escritorio. A partir de ahí, la navegación se produce recargando mediante AJAX los módulos correspondientes mediante sucesivas invocaciones. La estructura modular de la página incluye un módulo head y un módulo body, el cual se compone por un módulo menúPrincipal y un content. Dentro de ese content se carga un módulo para el usuario u otro: escritorio, estadísticas, carga, etc.

El módulo de menú sirve para dotar al sistema de un menú principal. En él existen tres opciones: perfil, gestión y estadísticas. En la opción de perfil un usuario puede ver sus datos personales (únicamente se utiliza nombre, email y teléfono). Puede cambiarlos a su gusto, así como su contraseña. En la opción de estadísticas se presenta la información en sí del equipo, con todos los datos cargados en el sistema. Existen diferentes vistas según la información que se quiera consultar, cargando el módulo correspondiente.

En la opción de gestión es donde entra en juego la diferencia por roles: un usuario normal sólo ve la página de bienvenida, mientras que un administrador accede a la gestión de cada tipo de dato. Puede añadir, editar o eliminar jugadores, jornadas, estadísticas, etc. Cada una de esta acciones se lleva a cabo invocando y cargando en pantalla su módulo correspondiente, que al final, como ya se ha dicho, consiste en un directorio que contiene el código php (con la estructura de ficheros mencionada). Para la página de bienvenida, además del primer mensaje que se presenta al entrar, consideré que sería positivo añadirle una sección con enlaces de interés. La estructura modular facilitó esta tarea, ya que era cuestión de añadir un módulo nuevo, similar al básico de la página primera (lo llamé gestion/escritorio), e incluirlo en el menú.

El código se estructura en tres directorios principales, modules, includes y js. En modules está la estructura de directorios necesaria para ejecutar y presentar cada uno de los módulos cargables por el plugin de documento, mientras que en includes tenemos clases de utilidades del Framework (por ejemplo las clases para trabajar con la base de datos, el sdk de Facebook, etc.), las clases envoltorio para manejar cada tipo de dato y el fichero de definiciones de variables globales. En js tenemos las librerías javascript que se ocupan de las interacciones con el navegador del usuario, como por ejemplo el efecto acordeón del menú lateral.

3.5. Implementación del login con Facebook

Una vez realizada la parte inicial de la aplicación, instalé el SDK PHP de Facebook e implementé las funcionalidad de login. Básicamente consiste en un intercambio de información entre servidores. Desde la página de login se realiza una petición HTTPS a una URI proporcionada por el SDK, a la cual se pasa como parámetro la url de callback a la que devolver la información. Si el usuario ya está logado en Facebook, se devuelve un token a la aplicación, con el cual se puede obtener la información del usuario mediante el SDK PHP.

Existen un par de condiciones previas que son necesarias para poder llegar a implementar esta funcionalidad. Imprescindible tener una cuenta de Facebook, primero, y registrar esa cuenta como cuenta de desarrollador después. Una vez dispones de una cuenta de desarrollador, puedes dar de alta aplicaciones. Disponen de una interfaz que hace las veces de panel de control de estas aplicaciones, y mediante él puedes obtener los identificadores de la aplicación y configurar las ULRs de callback de la misma. Estos datos son necesarios posteriormente para que funcionen las clases proporcionadas con el SDK.

3.6. Publicación de la aplicación

Para que los usuarios puedan acceder a la aplicación, esta debe ser visible, poder llegar a ella a través de una URL. En lugar de utilizar un hosting de pago tomé la decisión de servir la aplicación desde mi entorno local. Fue necesaria la configuración de la red interna de modo que, usando NAT, las peticiones a la IP externa de mi red local fueran redirigidas al ordenador, el cual obtenía su IP mediante DHCP. Al ser una red personal, con los mismos dos o tres equipos conectados siempre, era razonable asumir que siempre tendría la misma IP, al menos de manera regular. Esta configuración la realicé mediante la interfaz web del router de mi red local, aprovechando que los routers comerciales disponen de esta opción desde hace años.
Para el DNS utilicé la funcionalidad de la empresa No-ip [16]. Ofrecen hosting dinámico con hasta tres dominios gratuitos. En mi caso registré pfcequipo.no-ip.org y lo apunté a la dirección IP externa del router de mi domicilio. No se trataba de una IP fija, por lo que hice uso del software que provee No-ip, el cual corre un demonio que informa al servidor de los cambios de IP externa.

De esta forma se consigue tener el dominio apuntando al servidor de manera automática aunque la IP no sea fija.

Capítulo 4

Migración del código

Al retomar el proyecto, fue necesario reenfocarlo. Las versiones del servidor web y el motor php estaban obsoletas, por lo que centré la atención en adoptar una aplicación ya funcionando pero anticuada y modernizarla y adaptarla para versiones modernas. Elegí migrar el código a la versión 7.0 de PHP por ser estable ampliamente adoptada. Tuve que retocar las clases que se conectaban a base de datos dado que las funciones que utilizaba estaban ya *deprecadas* (obsoletas), no funcionaban con el motor nuevo. Así, rehice el *wrapper* de base de datos utilizando las nuevas funciones y retoqué todas las funciones que utilizaba.

El otro gran cambio que tuve que afrontar fue migrar el cargado de las clases. El motor en su versión antigua hacía uso de una función `__autoload()` que también se encuentra ya en desuso. De esto me di cuenta cuando reimplementé el login con Facebook, cuya versión también estaba obsoleta. Existen dos manera de utilizar el nuevo SDK que proporcionan, utilizando composer o instalándolo manualmente. Composer es un gestor de dependencias el cual no tenía ninguna seguridad de que fuera a funcionar correctamente en combinación con mi Framework, por lo que opté por la instalación manual, consistente en el copiado de las clases en una ruta del árbol de directorios de la aplicación y la inclusión de su propio autoload mediante la directiva `include` de PHP. Al hacer esta inclusión, se reescribía la función de carga de clases, por lo que las propias clases del Framework dejaban de poder utilizarse. Tuve que reescribir la función de autoload para adaptarla a las nueva necesidades. Una vez el login funcionó correctamente, me enfrenté a la lógica del enlazado de cuentas. La lógica que apliqué es la siguiente: intento recuperar el email del usuario que ha accedido (Facebook le comunica que tiene que dar permiso a la aplicación para obtener este dato al entrar por primera vez). Si el email está ya registrado en la base de datos de la aplicación, cargo el usuario correspondiente y lo utilizo como usuario actual de la sesión. Si

no, lo doy de alta, y a continuación lo cargo como usuario y se continúa con normalidad. Otra opción hubiera sido dirigir al usuario a la página de registro con los datos que se obtienen gracias al token de Facebook ya cargados, para que se diera de alta él mismo de manera efectiva. Me pareció peor solución y una experiencia de usuario más pobre, dado que si introduces tus credenciales en Facebook parece lógico esperar que vas a entrar a la aplicación sin dar pasos adicionales. Más tarde, probando, descubrí que el propio Facebook no siempre devuelve el email del usuario que se ha logado. En estos casos la inserción del nuevo usuario fallaría, por lo que decidí controlar el caso y mostrar un error al usuario. Dado que es algo que escapa del control de la aplicación (si Facebook no devuelve los datos del usuario no hay manera de identificarle) pensé que era mejor devolver un error, ya que el usuario siempre puede registrarse desde la página, sin pasar por Facebook.

Una vez realizada la reescritura del código, aproveché para instalar un nuevo servidor web más moderno, Nginx, el cual es ampliamente utilizado en la actualidad en detrimento de Apache (no ha sido abandonado pero sí ha disminuido su uso). Nginx es más ligero que Apache y su configuración me parece algo más limpia y sencilla.

Otra de las migraciones que tuve que hacer fue la relativa al dominio que utilizaba. Por algún motivo que desconozco la empresa NoIp dejó de ofrecer los dominios y subdominios del tipo xxx.no-ip.org como el que yo utilizaba, por lo que, mediante mi cuenta existente, di de alta uno nuevo, pfcequipo.hopto.org. El funcionamiento es exactamente igual que el anterior, pero la URL del dominio tuvo que ser ahora esa.

Capítulo 5

Dockerización

Una vez la aplicación corría correctamente en el entorno local, ya modernizada, llegó el momento de empaquetarla para su posterior despliegue en una máquina que hiciera las veces de servidor, con facilidad. La arquitectura consta de dos imágenes, una de base de datos y otra de servidor web con php. Como la filosofía de Docker es ejecutar un hilo en cada contenedor, era necesario tener dos diferentes para ajustarse a ella.

Para la imagen de base de datos partí de una imagen oficial de MySQL, obtenida directamente del hub de Docker. Creé mi propia imagen incluyendo una estructura de base de datos que ya tuviera las tablas y al menos unos datos básicos que mostrar. De este modo, al ejecutar un contenedor basado en esta imagen no se parte de cero, ya hay datos precargados. La tarea consiste en escribir un Dockerfile, que es un archivo de configuración donde se le indica a Docker qué acciones debe realizar para generar la imagen. Una vez realizado, se invoca la orden `build` de Docker, indicándole el Dockerfile a ejecutar y el nombre de la imagen a generar. Una vez construida la imagen, se utiliza la instrucción `run` para crear y lanzar un container de esa imagen. Para el caso del container de la base de datos, utilicé la opción –volume para indicarle un directorio del host donde se ejecuta, para de este modo tener persistencia de datos en caso de parada o error del container. Con –volume no se guardan los datos en el sistema de ficheros del container, si no que se realiza un enlace al directorio del host. Se puede parar, reanudar, recrear o eliminar el container y cualquier otro podrá recuperar los datos.

Para la imagen del servidor web utilicé como base una las numerosas imágenes disponibles en el hub de Docker. Escogí una con las versiones de Nginx y php que había utilizado en el desarrollo. A partir de ella, la reconfiguré para que sirviera las páginas de la aplicación mediante su correspondiente Dockerfile, creé la nueva imagen y ejecuté un container.

Para que se comunicaran entre ellos, utilicé la opción `--link` al ejecutar el servidor. Es decir, para ejecutar el container con el servidor web es necesario que esté corriendo el container con la base de datos previamente.

Una vez generadas las imágenes y comprobado su correcto funcionamiento en local, faltaba la parte de ejecutarlo en un entorno más estable, con servicio continuado. La opción por la que me decanté fue por registrarme en Google Cloud. Sus precios son muy competitivos y permiten un año de manera gratuita simplemente registrándote, con un crédito de 300 dólares a gastar durante ese año. La facturación la hacen en base a las instancias que se desplieguen, los recursos que se le asignen, las horas de uso, etc. Opté por crear una instancia de una máquina sencilla, utilizando las plantillas que proveen. En pocos minutos tenía una instancia de VM lista con un sistema operativo Ubuntu 16.04 corriendo. Elegí dotar a la instancia de una IP estática mediante el propio panel de control del Google Cloud, para ahorrarme dificultades con el DNS. La instancia se maneja mediante SSH, y haciendo uso de ello instalé Docker en ella. Para poder correr los contenedores de mis imágenes tenían que ser accesibles. El método que utilicé para ello fue registrarme como usuario del hub de Docker [3]. Con tu cuenta de Docker tienes gratuitamente espacio para tres repositorios, por lo que puedes dejar accesibles sin coste alguno hasta tres imágenes, cada una con sus diferentes versionados. Elegí hacer uno de ellos privado (sólo se puede hacer uso de la imagen si se conoce el usuario y la contraseña) por dos motivos: uno, para ver cómo funcionaba una imagen privada; y dos, para no dar libre acceso al código desarrollado. Subir las imágenes al hub es sencillo, basta con marcarlas con un tag opcional (es *latest* si no se indica nada) y hacer uso del comando `push` de Docker para subirlas al repositorio. En el caso del repositorio privado, es necesario identificarse mediante el comando login de Docker (`docker login`).

Ya con las imágenes disponibles, sólo restaba ejecutar los contenedores mediante el comando `run`, aplicando las opciones necesarias. Con esto la aplicación estaba en ejecución, disponible para accesos desde cualquier dispositivo del mundo con conexión a internet.

Como último detalle, apunté el DNS del dominio que tenía con NoIP a la IP estática de la instancia de VM de Google. Tras un breve tiempo durante el cual se expande la información del DNS, el sitio era ya accesible desde pfcequipo.hopto.org.

Capítulo 6

Securización del servidor

Con el sistema ya funcionando, surgió la mejora de proteger el servidor y servir las páginas mediante HTTPS en lugar de HTTP, ya que inicialmente utilicé la configuración básica de Nginx, la cual aplica al puerto 80 estándar. Esto hacía que las credenciales del usuario viajaran de manera insegura, en plano, pudiendo ser interceptadas sin dificultad. Es importante que la navegación sea segura, el usuario debe tener garantías, por lo que decidimos que lo mejor era que la aplicación funciona bajo TLS.

Para ello, hicieron falta tres pasos: generar un certificado, reconfigurar el servidor y rehacer la imagen de Docker. Para el primer paso me hice un certificado autofirmado utilizando OpenSSH, un software libre que permite la generación de certificados mediante comandos en la consola. El problema que da este tipo de certificados es que los navegadores no se fían de él al no estar reconocidos por una entidad certificadora superior (CA), de forma que no permiten navegar por el sitio web sin añadir excepciones manuales de seguridad. Para un aplicación orientada a ser utilizada más allá del ámbito académico sería mejor obtener un certificado emitido por una CA, pero en su mayoría son de pago y consideré que sería suficiente ilustrar la seguridad con un certificado propio, que además es muy sencillo y rápido de obtener. En la figura 6.1 se pueden ver los datos del certificado generado.

Una vez obtenido el certificado, reconfiguré el Nginx de mi máquina local para que redirigiera todo el tráfico HTTP a la versión HTTPS, añadiendo un bloque server al fichero de configuración que se ocupara de hacer la redirección de cada petición que le llegara al puerto 80, el archiconocido puerto estándar HTTP. También rehice la configuración para que hiciera uso de los archivos generados por OpenSSH, en este caso los .key y .cert que contienen las claves privada y pública; y que escuchara peticiones por el puerto

Figura 6.1: *Datos del certificado generado con OpenSSH*

443 estándar de HTTPS. Una vez verificado que se navegaba sin problemas, rehice la imagen de Docker del servidor, la subí de nuevo al hub y la ejecuté desde mi instancia virtual en el cloud de Google, parando el contenedor con la versión anterior. Con sólo un comando de la consola tenía la nueva versión securizada del servidor corriendo en pocos segundos.

Capítulo 7

Pruebas

7.1. Pruebas funcionales

Las pruebas han consistido, primero, en una validación funcional de la plataforma. En particular:

- login de usuario con rol de usuario usando el formulario
- login de usuario con rol de administrador usando el formulario: verificar acceso a los módulos no visibles en el otro rol
- login de usuario baneado: la plataforma no le deja acceder, el mensaje de error se lo indica
- registro de usuario nuevo
- edición de datos de un usuario: puede cambiar su nombre, email, contraseña, etc.
- logout de usuario
- login de un nuevo usuario desde Facebook: su usuario es creado en la base de datos y puede entrar
- login de un usuario existente desde Facebook: le reconoce y entra
- creación de una temporada
- edición de una temporada
- borrado de una temporada
- creación de una jornada

- edición de una jornada
- borrado de una jornada
- creación de estadísticas de una jornada
- edición de estadísticas de una jornada
- borrado de estadísticas de una jornada
- creación de un jugador
- edición de un jugador
- borrado de un jugador
- navegación por la página de estadísticas: se ven todos los módulos con la diferente información
- el orden en el que se presentan todas las estadísticas es de mayor a menor
- un administrador puede editar el texto de la página de bienvenida, un usuario no
- acceder a pfcequipo.hopto.org redirige automáticamente a https://pfcequipo.hopto.org

7.2. Pruebas de carga

Aparte, realicé unas pruebas de carga bombardeando al servidor de peticiones utilizando jmeter.
En primer lugar ataqué con 10 hilos, luego 100, luego 500 y finalmente 5000.
El informe de jmeter se recoge en las tablas 7.2 y 7.1.

Resultados de pruebas de carga con 500 hilos						
Muestras	**Media**	**Mín**	**Máx**	**% Error**	**Rendimiento**	**Kb/sec**
500	7185	1690	11096	0 %	44, 4	189

Cuadro 7.1: *Resultados pruebas de carga con 500 hilos*

La tabla 7.2 recoge los resultados acumulados de todas las ejecuciones, mientras que la 7.1 muestra una ejecución individual de 500 hilos.

Resultados de pruebas de carga acumuladas						
Muestras	**Media**	**Mín**	**Máx**	**% Error**	**Rendimiento**	**Kb/sec**
9048	16268	280	131895	23,674 %	0,49158	1,82

Cuadro 7.2: *Resultados pruebas de carga acumuladas*

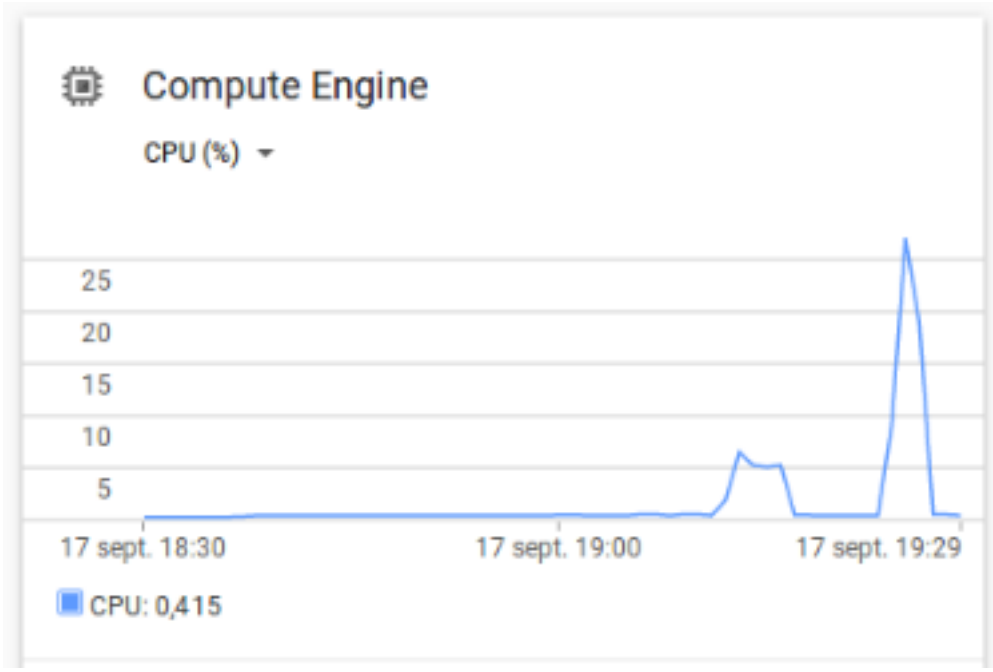

Figura 7.1: *Comportamiento del servidor en las pruebas de carga*

A la vista está que 500 hilos son manejados de sobra por el servidor, no hay errores y el rendimiento es muy alto.

Sin embargo, como se puede ver, al atacar con 5000 hilos se produce una media de un 23 % de error, en particular timeouts de la aplicación según los logs de jmeter. Los errores son de varios tipos:

- Non HTTP response code: java.net.ConnectException
- 502 Bad Gateway
- Non HTTP response code: java.net.UnknownHostException

Estos errores se deben a la configuración del servidor, que se ve desbordado por el número de peticiones. No se puede tratar de excesivo consumo de recursos en la máquina porque su uso de CPU en ningún caso llega siquiera al 30 %, como se ve en la figura 7.1.

Las pruebas con 10 hilos ni aparecen en la gráfica, y el uso de cpu supera el 25 % sólo cuando el servidor es atacado con 5000 hilos (el pico que se aprecia en la figura coincide con esta fase). La conclusión de la prueba es que la máquina esta **sobredimensionada**, no hacen falta tantos recursos. Es cierto que la carga podría ser algo mayor al acceder a la base de datos para mostrar las estadísticas, pero la aplicación está pensada para ser utilizada por un equipo amateur, por lo cual las estimaciones de tráfico no superarían las 15 o 20 sesiones concurrentes en un momento de pico (y de esas sesiones sólo una sería administrador, con capacidad de editar datos en la base de datos).
Las pruebas indican que el servidor está más que preparado para soportar el uso que se le pretende dar, y que sólo en caso de tráfico descontrolado (miles de peticiones) tendría problemas de respuesta.

También se comprueba que el servicio se recupera al parar el tráfico por sí mismo, no se pierde y hay que actuar manualmente para que funcione de nuevo, lo cual es positivo porque puede absorber picos inesperados puntuales. Se resolvería arrancando más containers y atacándolos mediante un balanceador de carga, o reconfigurando el servidor para aceptar más peticiones. Esto parece una peor idea dado que el tráfico medio esperado es mucho menor, y no es óptimo sobredimensionar el sistema.

Capítulo 8

Historia del proyecto

8.1. Primera aproximación al desarrollo

Cuando este proyecto se planteó la primera vez fue en 2012, y en 2013 programé el código y monté los primeros servidores desde el entorno local. En hacer el análisis de los requisitos y elegir y diseñar el modelo de datos tardé un par de tardes o tres. Después de eso, le tocó el turno a la programación de los módulos, partiendo de la base del framework preexistente. Partir de una base de este tipo tiene ventajas, ya que provee ciertos elementos, pero tiene también algún inconveniente, como el tiempo que supone la curva de aprendizaje para poder usarlo con facilidad. Al principio me costó un poco coger agilidad para editar los ficheros, conocer los flujos de información del sistema, saber dónde estaba cada elemento, etc. Sin embargo una vez controlada esa parte fue algo más sencillo implementar los módulos, dada su estructura similar. En cosa de cuatro o cinco semanas tenía los módulos que quería con la información que pretendía mostrar, contando también con el backoffice, la parte del sistema que se ocupa de la inserción y edición de datos. Al no haber hecho antes nada de diseño web, sí se me hizo duro utilizar los archivos CSS para una correcta visualización, era algo que no controlaba mucho y puede llegar a ser algo frustrante, dadas las diferencias entre navegadores, pantallas y demás. Antes de continuar con las funcionalidades añadidas (el enlazado de cuentas y la integración con Facebook), preparé la parte del DNS, registrando el dominio original (como se cuenta en la sección 3.6), configurando la red interna para hacer el servidor accesible y exponiendo la plataforma al exterior. Para realizar estas primeras pruebas instalé un servidor web apache y un motor php 5.3 en mi ordenador local. Una vez el código hacía lo que debía, me planteé el exponerlo al exterior. Fue una primera alegría el conseguir acceder a la plataforma desde un navegador web

sin pasar por localhost. Se podía navegar por las diferentes páginas, cargar la información, identificarte como un usuario sabiendo el sistema tu rol...era la base del sistema.

8.2. Los problemas de integrar Facebook

Con estos primeros pasos ya asentados, procedí a avanzar con la parte de Facebook. Lo primero fue hacer una cuenta en su plataforma, ya que no era usuario previamente. Decidí no usar datos personales, di de alta una dirección de correo de hotmail, pfcequipo@hotmail.com, y la usé como email para el registro en Facebook. La verdad que no tuve ningún problema para darme de alta, incluso sin poner ningún tipo de dato personal, foto alguna o conexión con nadie. Pasé entonces a registrarme como desarrollador con la cuenta recién creada, para poder crear la aplicación necesaria para poder hacer el login. Para ello solicitaban en su momento un número de teléfono para poder enviar un SMS de seguridad con un código a introducir. No tuve otro remedio que, ese sí, poner un dato real. Usé mi número personal y el mensaje llegó más de dos semanas en llegar. Este retraso imprevisto es algo que no se puede descartar siempre que se trabaje con sistemas de terceros. Es muy dificil de prever y controlar, pero cualquier desarrollo de este tipo está expuesto a que le ocurra, incluidos los del ámbito puramente profesional. Un retraso de estas características puede hacer que se retrasen las fechas de entrega con un alto coste económico para la empresa que lo sufra. En cualquier caso, cuando llegó el código no tuve más problema, su interfaz es sencilla de utilizar y a través de ella obtuve los datos necesarios para continuar.

Así, creé la aplicación en modo test, configurándola para que solicitara permisos de acceso al email del usuario (no es uno de los permisos por defecto). En este modo de pruebas sólo deja acceder al creador y las cuentas que le de acceso, no está abierta al público en general. Me pareció una buena idea por estar en pleno de proceso de desarrollo, pero en realidad lo único que me aportó fue dificultad para realizar las pruebas. Tuve que crear una cuenta falsa extra que nunca llegué a utilizar, no podía pedir ayuda fácilmente a usuarios habituales de Facebook. Ahora veo que no hubiera pasado nada por abrirla desde el principio, tampoco es que nadie se fuera a interesar por ella sin anunciarla, darle visibilidad.

Una vez obtenidas las claves de la aplicación necesarias para hacer el login, primero lo intenté con su API Javascript. Para ello, en principio, había que añadir un script que ellos proporcionaban y que se encargaba de gestionar las peticiones a sus servidores y ver las respuestas. Me resultó confuso y complicado porque no llegué a conseguir utilizarlo para identificarme y acce-

der, el navegador bloqueaba las peticiones por problemas de cross domain. Nunca llegué a saber qué ocurría exactamente, porque intenté la otra opción, hacer el login con el SDK PHP, y esa opción fue mejor. En aquel momento resultaba algo extraña, ya que en el navegador, en lugar de un pop-up o formulario donde introducir tus credenciales, se producía una redirección a una URL del propio Facebook, para luego regresar a la plataforma mediante la llamada de su servidor a la URL de callback proporcionada. Hoy es de lo más habitual trabajar de esta manera, multitud de páginas lo hacen así, desde la imposición de OAuth es mucho más común ver este tipo de redirecciones para luego volver. Desde diarios online como elpais.com hasta grandes plataformas de ecommerce como booking.com, pasando por plataformas de distribución como justeat.es utilizan ya a día de hoy login mediante cuenta de Facebook haciendo uso del SDK PHP. Tardé un par de semanas más en conseguir procesar adecuadamente la llamada de Facebook con la identificación del usuario. Tuve que refactorizar la clase que se ocupaba de la gestión del login para adecuarla a esta nueva opción, y las pruebas de integración no me resultaron sencillas, al ser la aplicación privada en Facebook y tener todo el sistema ejecutando en local. Cuando conseguí tener la información que necesitaba capturada y procesada, tuve que tomar la decisión de qué hacer con los usuarios que entraban sin tener cuenta en el sistema, sólo utilizando su cuenta de Facebook (lo que es al final el enlazado de cuentas en sí). Opté por redirigir al usuario a la página de registro, estando los campos rellenos automáticamente con la información recabada desde el API. De esta forma, el propio usuario se daba de alta en el sistema, a efectos prácticos. En las pruebas me pareció una experiencia de usuario un poco pobre, porque obligaba a hacer una cuenta en el sistema cuando como usuario ya habías dado permiso para poder utilizar la de Facebook para identificarte. En ese momento, además, necesitabas volver a logarte para acceder a la plataforma tras registrarte, lo que hacía la experiencia de usuario un poco peor aún.

Una vez acabada esta parte, ocupé otro par de días más en utilizar otro método del API de Facebook para realizar autopublicaciones. La idea era que cada vez que un administrador creara una jornada nueva, es decir, se añadiera información de interés para los usuarios, se generara una publicación en el muro de la aplicación en Facebook, que idealmente estarían siguiendo esos mismos usuarios.

8.3. El reenfoque de la aplicación

En ese momento dimos por buena la aplicación y comenzó el momento de la redacción. Pasado el tiempo decidimos darle el nuevo enfoque de la actua-

lización del código y el uso de Docker para los despliegues. Esta nueva parte del proyecto la empecé montando un nuevo entorno. Todo el desarrollo anterior lo hice sobre mi antiguo ordenador personal con un Ubuntu 10 que nunca actualicé para no poner en peligro el funcionamiento de la aplicación. Para el nuevo entorno utilicé un portátil Acer con Windows 10 sobre el que monté máquinas virtuales usando Oracle Virtualbox. Este software de máquinas virtuales me permitió ejecutar de manera segura sistemas operativos diferentes para hacer pruebas.

Uno de mis mayores miedos todo este tiempo fue que el ordenador original dejara de funcionar, ya que sabía de la dificultad que supondría replicar el entorno. Esto da una idea de cuán importante es la posibilidad que ofrece Docker de funcionar independientemente del entorno, ya que si hubiera que desplegar la aplicación en las condiciones originales pero en otra máquina no sería una tarea nada sencilla. De hecho, lo intenté replicar desde cero con una máquina virtual, pero no lo conseguí, no es sencillo instalar la versión de PHP tan antigua que yo utilizaba (5.3) ya que está totalmente obsoleta y, por ejemplo, no está ya en los repositorios oficiales, con lo que la instalación adquiere mayor complejidad. Lo que sí es sencillo es instalar una versión actualizada de Ubuntu con el software necesario para ejecutar la aplicación. Añadirle una base de datos MySQL, un Nginx y un motor PHP nuevo también se consigue de una manera bastante directa instalándolos con sudo `apt-get install`, es casi trivial con Ubuntu [2]. Por tanto, parecía una mejor elección migrar el código y hacerlo funcionar en entornos modernos. Me llevó casi dos semanas migrar el código para que PHP7 consiguiera ejecutarlo sin fallos. Tuve que sustituir las funciones de conexión y uso de base de datos anticuadas por las nuevas, modificando las clases que hacen de *wrapper* (envoltorio, una forma de encapsular la funcionalidad necesaria en unas pocas clases útiles). Básicamente consiste en orientar a objetos antiguas funciones procedimentales, lo cual encaja mucho mejor con el modelo de clases utilizado por el sistema: es la forma más lógica y limpia de proceder. Tras ello, tuve que modificar el encabezado del código PHP incrustado en los HTML definidos en los archivos .tpl.php. En la versión anterior, se podía indicar al motor que debía ejecutar un código PHP antes de devolver el HTML escribiéndolo entre símbolos `<? ?>` , pero en esta nueva versión el código debe estar incluido entre los marcadores `<?php ?>` Creé un script que introdujera los caracteres en los lugares adecuados y la página comenzó a visualizarse correctamente. Lo complicado no fue cambiar el código, fue saber por qué la página no se veía sin ello. Para más información sobre la migración, remitirse al capítulo cuatro donde se encuentran los detalles.

8.4. Cómo migrar el sistema y adecuarlo a su uso con Docker

Con la migración más o menos realizada, me encontré que no se podía navegar por la aplicación, el servidor devolvía errores 502 de una manera que daba la impresión de ser arbitraria. Lo corregí añadiendo dos directivas a la configuración del Nginx, aumentando el tamaño de los buffers del PHP. Para encontrar el error investigué los logs del servidor, que es la herramienta básica a la que se debe acudir cuando algo no está funcionando correctamente. La información que ofrecen depende del nivel de log configurado, y en ocasiones ésta no es sencilla de interpretar, pero es el lugar al que acudir para comenzar a buscar las causas de los errores. A continuación me puse a trabajar para crear las imágenes de Docker. Sufrí un contratiempo ya que la máquina virtual que estaba utilizando se quedó sin memoria al hacer pruebas con Docker, por lo que tuve que crear otra con más memoria asignada para no quedarme corto (10Gb contra 25Gb). Me llevó más o menos otra semana aprender los conceptos básicos de Docker y generar mis propias imágenes. Cuando estuve listo, ejecuté los containers y verifiqué que la aplicación funcionaba en el entorno local, momento en el cual me enfrenté a la manera de desplegar la aplicación en otro entorno. Estuve un par de días valorando opciones y me decanté por Google Cloud. Antes de crear el nuevo entorno en la nube, tuve que decidir cómo hacer que ese entorno fuera capaz de acceder a mis imágenes de Docker. Me planteé la opción de crear mi propio repositorio de imágenes (Google ofrece una herramienta para ello), pero finalmente opté por usar el hub de Docker, que es más estándar y está configurado por defecto. Tuve un malentendido porque ellos utilizan el concepto repositorio de una manera distinta a la más habitual: pensé que para ellos un repositorio era un sitio donde guardar tus imágenes, pero no, en cada repositorio sólo puedes guardar una. Lo llaman repositorio porque de esa imagen puede albergar todas las versiones que se deseen, marcándolas con tags para diferenciarlas. Con esto perdí una tarde, pero al final las imágenes estaban listas para ser utilizadas donde fuera.

Crear la instancia de máquina virtual en el Google Cloud fue sencillo, en un par de horas tenía todo lo necesario. La instancia estaba creada con su IP fija, sus recursos asignados y el software necesario (Docker es lo único que instalé en la instancia, y un cliente mysql para hacer pruebas). Ejecutando los `docker run` y cambiando el DNS en noip.org la aplicación por fin estaba accesible para el público.

Después de este hito me puse a migrar la parte de Facebook. Esto me llevó más de una semana de trabajo. Como ya he explicado anteriormente,

al instalar el SDK de Facebook sobrescribí la función de `__autoload()`, por lo que las clases de la aplicación dejaron de funcionar. Tuve que migrar esta parte también, creando la versión actualizada que hace uso de la carga más moderna, con la función `spl_autoload_register()`. Con esto completé la migración. También decidí modificar cómo funcionaba la aplicación: desde ese momento todo usuario que entrara utilizando el login de Facebook sin tener cuenta previa sería dado de alta en la tabla usuarios con los datos recogidos de su sesión. De ese modo accede directamente a la aplicación y la experiencia es más fluida. Cuando pensaba que todo estaba correcto, descubrí que no todo el mundo podía entrar con su cuenta de Facebook. De algunos usuarios no llegaba el email por lo que el enlazado de cuentas no se podía realizar correctamente, y para estos casos decidí que lo mejor era capturar el error e indicárselo al usuario, ya que escapa al control del sistema esta falta de datos.

8.5. La problemática de conseguir una navegación segura

Ya por último le llegó el turno a la securización del servidor. Hasta ese momento toda la aplicación se servía usando HTTP. La herramienta elegida para crear los certificados necesarios fue OpenSSH, por su gratuidad y rapidez. Intenté también usar el certbot de Let's Encrypt pero obtenía un error al intentar generar los certificados. Al ser un proyecto académico decidí continuar con el certificado autofirmado, aunque los navegadores no se fiaran de él. Rehacer la configuración del servidor para funcionar por HTTPS me llevó en total otros tres o cuatro días, contando el rehacer las imágenes porque hice pruebas con imágenes distintas a la que había elegido como base, por si fueran más ligeras y sencillas. Al final me quedé con la original porque tenía todo lo que necesitaba, mientas que otras requerían de ampliaciones y pruebas a hacer con el Dockerfile de creación. Volví a tener el problema del 502 arbitrario, por lo que tuve que hacer de nuevo unos ajustes en la configuración del servidor. Finalmente, cuando todo funcionaba en local, volví a subir la nueva imagen al hub de Docker, esta vez con un nuevo tag :ssl y a ejecutar un nuevo container en la instancia del cloud.

Capítulo 9

Conclusiones y trabajos futuros

En este Proyecto se ha llevado a cabo el desarrollo, mantenimiento y mejora de una aplicación a lo largo del tiempo. Se ha analizado y explicado (incluidas las dificultades encontradas) el proceso para:

- Diseñar y desarrollar una aplicación web.
- Trabajar con frameworks modulares desarrollados previamente.
- Integrar la aplicación con el API de un servicio externo.
- Actualizar legacy code a nuevas versiones de su lenguaje.
- Usar sistemas de despliegue y seguridad modernos.

La conclusión principal de este proyecto es que los objetivos planteados en el capítulo 1.2 eran adecuados y realizables.

Además, se puede mencionar que resulta razonable y positivo migrar aplicaciones antiguar a los modelos nuevos. Las tecnologías evolucionan para ofrecer soluciones a los problemas, y aunque puede que introduzcan algo de complejidad al final compensa por las ventajas que ofrece. Por ejemplo, hubiera sido complicado distribuir la aplicación y tedioso montar los entornos necesarios para que funcionara en diferentes sistemas con el modelo inicial. Sin embargo, al utilizar Docker, el proceso de despliegue se hace trivial, sin más que ejecutar las dos imágenes. Se independiza del entorno (sólo hace falta que Docker funcione) y se hace mucho más sencillo de gestionar. Puede parecer que hacer modificaciones sobre el código se hace más complicado, ya que cualquier mínimo cambio supone rehacer la imagen, subirla de nuevo al hub, bajarla donde sea necesario y relanzar el container nuevo. Sin embargo, imaginemos que el código estuviera en dos servidores diferentes: cada cambio habría que hacerlo manualmente en ellos, lo cual es propenso a errores

humanos, olvidos, etc. Esta nueva manera de hacer las cosas nos pone en las puertas del *continuous delivery*: es posible automatizar las acciones mecánicas (rehacer la imagen, subirla de nuevo y demás) y los despliegues de código con herramientas como Jenkins, con la cual podríamos subir de una manera muy cómoda cada cambio de código que se haga en local al servidor, con plena confianza en su funcionamiento (una vez validado, claro).

9.1. Objetivos cumplidos

Los objetivos del proyecto se pueden resumir en estos cuatro puntos: analizar y plantear una aplicación, desarrollarla de manera modular, integrarla con Facebook y modernizarla con Docker.

- Análisis de la aplicación. Este objetivo está cubierto por la parte de la toma de requisitos y planteamiento de la aplicación. Estudié qué datos eran atractivos para un posible usuario hablando con mis compañeros de equipo, diseñé una manera de presentarlos y me enfrenté a un diseño desde cero de una aplicación de complejidad aceptable. Tomé la decisión de utilizar un Framework para reaprovechar funciones cuyo funcionamiento estaba probado, para así optimizar el aprovechamiento de recursos. Hacer software no es reinventar la rueda cada vez, aprovechar lo que ya funcione es un gran ahorro de tiempo y de dinero.

- Desarrollo Modular. El uso del framework elegido hizo posible este desarrollo en módulos, y cada apartado de la aplicación está codificado y desarrollado como uno de ellos. De esta forma resulta sencillo añadir funcionalidades nuevas, mediante el uso de nuevos módulos que encajen con los ya existentes. Al ser independientes, se puede agilizar el desarrollo.
  La migración a PHP7 también ha sido muy positiva: hace el código más legible al estar más orientado a objetos y al estar actualizado evita posibles bugs que se arreglan con el paso de las versiones, los motores van más rápido y consumen menos recursos, lo cual es bueno para el sistema porque está menos exigido; y da pie a ampliaciones que se hagan sobre la base de lo que vaya aportando de nuevo el lenguaje, en su evolución. Sin migrar no se podría utilizar funciones nuevas que salgan a la luz.

- Integración con Facebook. Dado que ambos sistemas se comunican correctamente, este objetivo ha sido también cubierto. Esta integración permite la federación de la identidad del usuario, ofrece una experiencia más fluida y facilita el uso, ya que elimina la necesidad de registro, la cual a muchos usuarios le resulta incómoda.

- Uso de Docker. La aplicación está recogida en dos imágenes, las cuales están disponibles en el hub de Docker (eso sí, una de ellas es privada por lo que requiere de contraseña para poder ser descargada y utilizada). Por tanto, el objetivo del uso de esta tecnología ha sido también satisfecho, el despliegue de la aplicación es ahora muy sencillo. La aplicación es escalable y se puede desplegar en diferentes entornos sin necesidad de configuración o ajustes, lo cual es un avance significativo respecto al modelo clásico de despliegue basado en servidores de aplicaciones.

9.2. Objetivos incumplidos

Como objetivo incumplido podría citar la migración de la autopublicación en Facebook. Por algún motivo que desconozco el mismo código que funciona para controlar el login (crear un objeto de la clase Facebook/Facebook y usarlo para la gestión de las peticiones y repuestas entre los sistemas) no funciona para publicar contenido. Dado que el plazo se acaba y que es una funcionalidad secundaria creo que no es grave, pero hubiera sido un *nice to have*. Supongo que la solución pasaría por revisar la carga de clases de la plataforma, subiendo el nivel de traza del motor PHP para tener acceso a más datos de qué es lo que no consigue hacer. A mi entender, esto llevaría varias horas de trabajo.
Otro objetivo que cabe citar aquí hubiera podido ser generar un certificado firmado por una entidad certificadora, como por ejemplo Let's Encrypt, como se comenta en el capítulo de Historia del proyecto. Sólo la falta de tiempo me llevó a no investigar y solventar la causa del error, que en principio pasaría por reconfigurar el servidor web NGinx de forma que tuviera una configuración acorde a lo que el software que provee Let's Encrypt (cerbot) espera encontrarse. Creo que sería algo relativamente sencillo, pero que impliaría también rehacer la imagen de Docker y redesplegarla.

9.3. Trabajos y ampliaciones futuras

Con más tiempo hubiera atacado muchos frentes. A continuación detallo algunos de ellos:

- El más importante, creo, la actualización del diseño y la adaptación a dispositivos móviles. El diseño ha quedado en cierto modo anticuado, sería muy positivo darle un nuevo aire, remaquetar las vistas, modernizarlo. La página debería ser *responsive*, es decir, adaptarse a diferentes

tamaños de pantalla en función del navegador que la solicite, para así mejorar la experiencia de usuario en smartphones y tablets.

- Otro punto sería reforzar la robustez, introduciendo más control de los datos introducidos por los usuarios. En principio la página está orientada para que sólo los administradores puedan crear o editar datos, y se supone que este tipo de usuarios sabe utilizar la herramienta; pero aún así un mejor control de los datos introducidos aportaría seguridad y controlaría errores.

- Otra mejora que me parece interesante sería ampliar el modelo de datos para dar cabida a las estadísticas también de los rivales. De esta forma se podría llevar el control de una liga entera, no sólo de un equipo. Además, datos como rebotes, asistencias, tapones, robos, faltas recibidas...hay muchas más estadísticas que se podrían gestionar. Para ello sería necesario sacar esos datos de alguna parte, por ejemplo anotando durante los partidos con una aplicación de terceros, o a mano. En último término, puestos a pensar a lo grande, se podría hacer una app para Android o iOS que sirviera para tomar las estadísticas en vivo e inyectarlas en la base de datos directamente a través de un API. Al no disponer de estos datos por no aparecer en las actas oficiales, no los metí en el proyecto, pero si creciera en el futuro sería algo deseable.

- Desde el punto de vista de la arquitectura y la gestión del sistema, podría investigarse el uso de Docker Compose para el tratamiento de los contenedores Docker. Se trata de una herramienta que permite combinar uno o más contenedores, ejecutándolos de la mano, relacionados mediante un fichero .yml. Esto simplificaría aún más la administración del sistema ya que la instalación y despliegue quedaría reducido a la ejecución de un único que comando que interpreta un fichero con las instrucciones necesarias.

- Por otra parte, también se podría avanzar en la integración de otras redes sociales, como Twitter, Google+ o Linkedin. Hay infinidad de sitios web que permiten al usuario identificarse mediante su cuenta de Google, y el método es casi igual al que he seguido para Facebook. Otra opción es tuitear contenido directamente en Twitter con un botón, sugerir en Linkedin, etc.

Apéndice A

Manual de instalación

Para instalar la aplicación es condición indispensable disponer de una máquina en la que instalar Docker. Hay algunos funcionamientos extraños reportados en sistemas operativos de Windows, por lo que yo he utilizado un Ubuntu, la versión LTE 16.04. Tras instalar Docker, sólo es necesario ejecutar los contenedores:

```
 >sudo docker run -d --name=bbddequipo --volume=/directorio
Elegido:/var/mysql/data ddosantos/mysqlequipo
 >sudo docker run -d --name=serverssl -p 80:80 -p 443:443
--link  bbddequipo:bbddequipo ddosantos/serverequipo:ssl
```

Se puede comprobar que los containers se han ejecutado correctamente con

```
 >sudo docker logs [nombreContainer]
```

El servidor estará escuchando peticiones en los puertos 80 y 443.

Apéndice B

Manual de usuario

Lo primero es acceder a pfcequipo.hopto.org y añadir una excepción de seguridad para que el navegador permita continuar. Así se llega a la página de login, representada por la figura B.1.

Desde la página inicial hay tres opciones: entrar con credenciales propias del sistema, entrar con Facebook o registrarse.

- Entrar con credenciales propias: introducir usuario y contraseña en los campos correspondientes y pulsar el botón **Entrar!**
- Entrar con cuenta de Facebook: pulsar la imagen **Login with Facebook**, la cual llevará a la página de entrada correspondiente https://www.facebook.com/login.php. Una vez introducidas las credenciales, el navegador vuelve a la aplicación con el usuario ya reconocido.
- Registrarse: pulsar el botón **Registrarse**, rellenar los datos del formulario (ver B.2) y completar la acción con el botón **Registrar**. Si todo es correcto, se presenta un página con un mensaje de OK y un botón para volver a la home, desde donde se puede acceder con las credenciales recién creadas.

Una vez dentro de la aplicación, lo primero que ve el usuario es el módulo *Escritorio*, con un mensaje de bienvenida (B.3).

Para ver las estadísticas, pinchar en el menú sobre la opción correspondiente *Estadisticas*. Se ve una tabla resumen con los líderes de cada categoría y la izquierda un menú para elegir opciones más específicas (ver B.4). Se puede elegir la que se desee y ver los datos en el módulo principal, como por ejemplo se representa en la figura B.5.

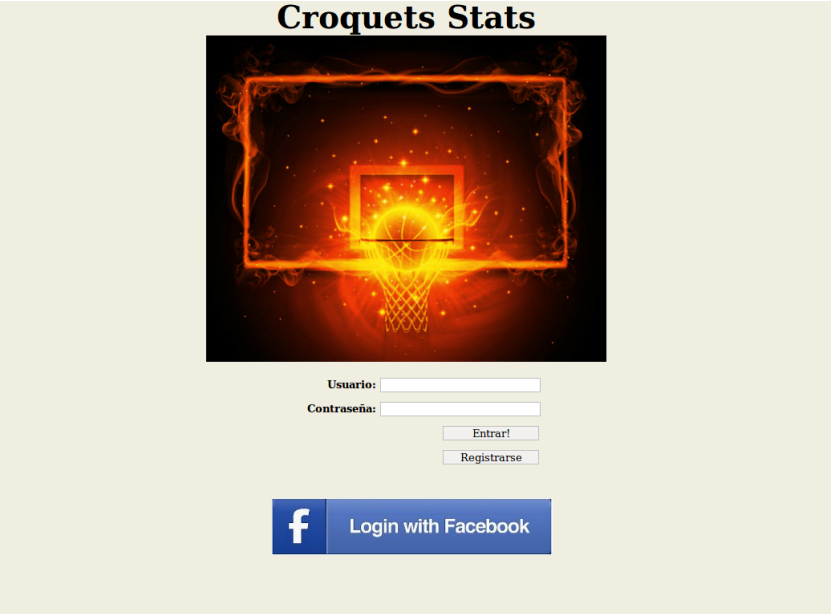

Figura B.1: *Página de inicio*

Para gestionar datos, lo primero es entrar con unas credenciales de administrador. Después, desde la página de bienvenida, se selecciona en el menú de la izquierda *Administración* y se despliegan los tipos de datos que se pueden editar, como se aprecia en la figura B.6.

Seleccionando cada uno de ellos se accede al módulo correspondiente para añadir, editar o borrar el elemento deseado. Por ejemplo, para datos de jornadas, ver B.7.

Para editar los datos del usuario actual o salir, seleccionar en el menú principal la opción *Perfil*. Se accederá al módulo correspondiente, como se puede ver en las figuras B.8 y B.9.

Figura B.2: *Página de registro*

Figura B.3: *Página de bienvenida*

Figura B.4: *Página de resumen estadístico*

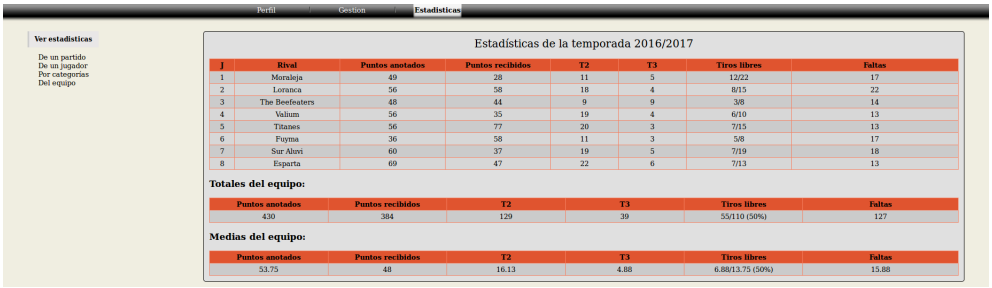

Figura B.5: *Página de estadísticas del equipo*

Figura B.6: *Página de inicio con credenciales de administrador*

Figura B.7: *Página de gestión de jornadas*

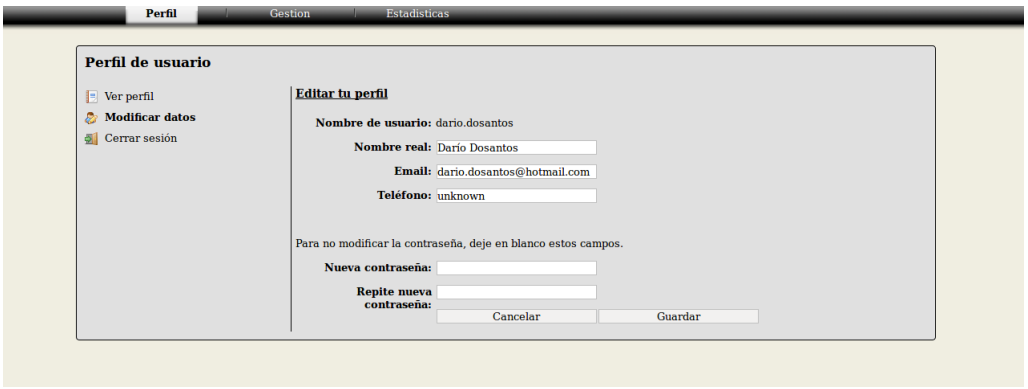

Figura B.8: *Página de edición de datos del usuario*

Figura B.9: *Página de perfil*

Apéndice C

Presupuesto del proyecto

En este capítulo he intentado estimar lo que hubiera costado realizar este proyecto en el ámbito profesional. Es perfectamente razonable pensar que este encargo se le pudiera hacer a un desarrollador, por lo que he tratado de ajustar lo que hubiera podido presupuestar.
El coste principal es el dinero a invertir en el personal, ya que el software utilizado es en su totalidad código libre y no es necesaria licencia alguna. Se representa este coste en la tabla C.2.
El único gasto material sería la máquina donde realizar los desarrollos y las pruebas, la luz y la conexión a internet, pero no es algo que en el ámbito profesional se cobre al cliente, lo suele proporcionar el desarrollador. Sí me parece interesante indicar el gasto de mantenimiento de la aplicación: el tenerla desplegada en una instancia virtual de Google sí genera gastos mensuales, que se pueden encontrar en la tabla C.3.

Fases del proyecto		
Fase 1	*Análisis y diseño*	40 *horas*
Fase 2	*Desarrollo del software*	240 *horas*
Fase 3	*Integración con Facebook*	40 *horas*
Fase 4	*Migración del código*	80 *horas*
Fase 5	*Dockerización*	50 *horas*
Fase 6	*Securización y despliegue*	16 *horas*
Fase 7	*Documentación*	60 *horas*
Total		526 *horas*

Cuadro C.1: *Fases del Proyecto*

En la Tabla C.1 especifico las fases del proyecto y el tiempo aproximado dedicado a cada una de ellas. Creo que es harto complicado determinar el

número de horas dedicadas, el valor se debe considerar estimativo. Asumiendo jornadas de ocho horas al día cinco días a la semana, intento contabilizar estas horas están y cuantificar su coste en la tabla C.2. En ella la figura Hombre-mes equivale a 131, 25 horas, lo que hacen semanas de 40 horas trabajadas.

Coste directo por personal			
Categoría	**Dedicación (hombres-mes)**	**Coste (€/hombre-mes)**	**Importe (€)**
Analista programador	4	2954, 26	11817, 04
Total			8862, 78

Cuadro C.2: *Costes directos de personal*

En la Tabla C.3 pongo como ejemplo una factura real emitida por Google por el servicio prestado de alojamiento, correspondiente al mes de agosto. Al estar en período de prueba descuentan el 100 % de la factura, pero pasado un año habría que pagar por el servicio.

Coste mensual del servidor			
Descripción	**Cantidad**	**Unidad**	**Importe (€)**
Compute Engine Standard Intel N1 1 VCPU running in Frankfurt	297,933	Horas	15, 59
Compute Engine Cargo de IP estática sin utilizar	298,581	Horas	2, 54
Compute Engine Storage PD Capacity in Frankfurt	10,033	Gibibyte-months	0, 41
Compute Engine Sustained Usage Discount			−1, 18
Total			17, 36

Cuadro C.3: *Coste mensual del servidor*

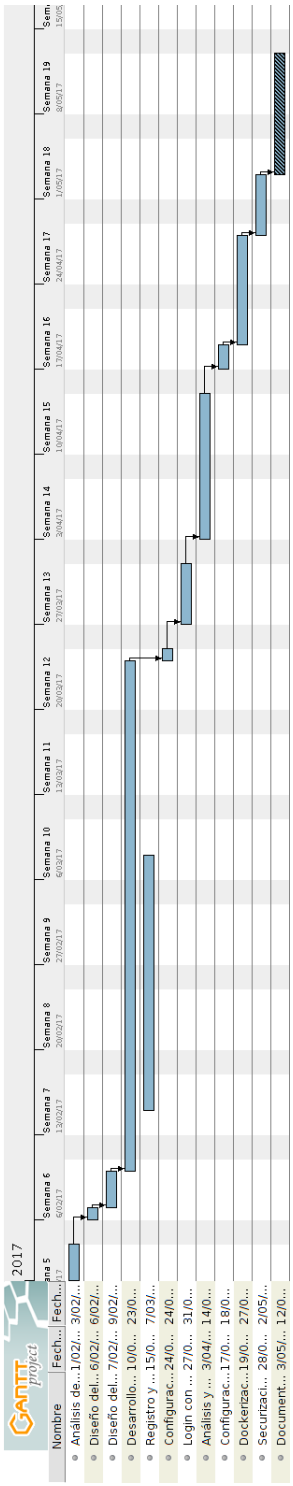

Figura C.1: *Diagrama de Gantt del proyecto*

Bibliografía

[1] Tim Berners-Lee. The world wide web project. `http://info.cern.ch/hypertext/WWW/TheProject.html`, 1990. [Online; accedido 18-septiembre-2017].

[2] DigitalOcean. Cómo instalar linux, nginx, mysql, php (lemp stack) in ubuntu 16.04. `https://www.digitalocean.com/community/tutorials/como-instalar-linux-nginx-mysql\\\-php-lemp-stack-in-ubuntu-16-04-es`, 2016. [Online; accedido 02-agosto-2017].

[3] Docker. Docker hub. `https://hub.docker.com/`, 2017. [Online; accedido 19-agosto-2017].

[4] Docker. What is a container. `https://www.docker.com/what-container`, 2017. [Online; accedido 19-agosto-2017].

[5] Paul DuBois. *MySQL*. Addison-Wesley Professional, 2013.

[6] Facebook. Primeros pasos - sdk para web. `https://developers.facebook.com/docs/php/gettingstarted`, 2017. [Online; accedido 06-agosto-2017].

[7] Facebook. Uso de la api graph. `https://developers.facebook.com/docs/graph-api/using-graph-api`, 2017. [Online; accedido 06-agosto-2017].

[8] W.Jason Gilmore. *Beginning PHP and Mysql5*. Apress, 2006.

[9] A2 Hosting. The top 5 well known businesses that use joomla. `https://www.a2hosting.com/blog/top-5-well-known-businesses-use-joomla/`, 2016. [Online; accedido 19-agosto-2017].

[10] Ecma International. Standard ecma 262. https://www.ecma-international.org/publications/standards/Ecma-262.htm, 1999. [Online; accedido 18-septiembre-2017].

[11] ISO. Iso 16262. https://www.iso.org/standard/55755.html, 1999. [Online; accedido 18-septiembre-2017].

[12] Joomla. Joomla! https://www.joomla.org/about-joomla.html, 2005. [Online; accedido 19-septiembre-2017].

[13] Marketing4ecommerce. El número de usuarios de internet en el mundo alcanza el 50 % de la población. https://marketing4ecommerce.net/usuarios-de-internet-mundo-2017/, 2017. [Online; accedido 10-septiembre-2017].

[14] Julie C. Meloni. *PHP, Mysql y Apache*. Anaya, 2009.

[15] Nanotutoriales. Cómo crear un certificado ssl de firma propia con openssl y apache http server. https://www.nanotutoriales.com/como-crear-un-certificado-ssl-de-firma-propia\-con-openssl-y-apache-http-server, 2013. [Online; accedido 04-septiembre-2017].

[16] NoIp. Free dynamic dns. http://www.noip.com, 1999. [Online; accedido 18-septiembre-2017].

[17] OAuth. Definitions: Oauth 2.0. https://www.oauth.com/oauth2-servers/definitions/, 2017. [Online; accedido 03-septiembre-2017].

[18] PHP. Php: releases. http://php.net/releases, 2001. [Online; accedido 23-agosto-2017].

[19] PHP. Php: Resumen de las funciones de la extensión mysqli. http://php.net/manual/es/mysqli.summary.php, 2017. [Online; accedido 12-agosto-2017].

[20] PHP. Php: spl_autoload_register. http://php.net/manual/es/function.spl-autoload-register.php, 2017. [Online; accedido 23-agosto-2017].

[21] Laura Stamey. Docker's tools of mass innovation: Explosive growth from open-source containers to commercial platform for modernizing and managing apps. http://www.

hostingadvice.com/blog/dockers-explosive-growth-from-\
\open-source-containers-to-commercial-platform/, 2017. [Online; accedido 10-septiembre-2017].

[22] W3C. Ajax introduction. https://www.w3schools.com/xml/ajax_intro.asp, 2005. [Online; accedido 19-septiembre-2017].