Resumen

onLeague es una aplicación web de simulación de una liga de fútlbol, más conocida como liga fantástica, que surge de la gran acogida de este tipo de juegos en otros países. Los usuarios pueden gestionar su propio equipo de fútbol, con el objetivo de conseguir el mayor número posible de puntos con las actuaciones de sus jugadores en los partidos reales. Mediante la aplicación de técnicas de desarrollo ágil se ha conseguido un proceso muy dinámico y flexible, permitiendo la introducción de cambios en los momentos necesarios. Asimismo, el desarrollo dirigido por pruebas ha proporcionado una mayor estabilidad a la hora de desplegar la aplicación en un entorno en producción. La evolución del proceso en iteraciones, junto con este entorno de producción en constante funcionamiento, ha permitido la interacción directa con usuarios. Por último, el uso de tecnologías ágiles y de código abierto como *Ruby on Rails*, ha conseguido incrementar la productividad general del proceso y minimizar el gasto económico. *onLeague* es un ejemplo de juego de simulación vía web que pone de manifiesto las ventajas del uso de metodologías de desarrollo ágil en un entorno de producción con usuarios reales.

Palabras clave: *onLeague*, liga fantástica, simulador de fútbol, desarrollo ágil, programación extrema, desarrollo dirigido por pruebas, integración continua, *Ruby on Rails*, *Ruby*.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

onLeague es una aplicación web de simulación de una liga de fútbol, más conocida como liga fantástica. En este tipo de juegos los usuarios pueden crear sus propios equipos, basándose en los jugadores de una liga de fútbol profesional. Cada jugador tiene su propio precio y con un presupuesto limitado los usuarios tienen que confeccionar un equipo y gestionarlo cada jornada. Este equipo irá consiguiendo puntos según el comportamiento de los jugadores en la vida real y el objetivo de los usuarios será alcanzar el mejor puesto posible en una clasificación general de equipos.

La idea del juego surge de la gran acogida que tienen hoy en día este tipo de ligas fantásticas en otros países y otros deportes, a diferencia del nuestro, en el que aun existiendo una amplia difusión del fútbol no están tan extendidas. En España existen algunas ligas fantásticas de fútbol, pero son muy comerciales o demasiado cerradas, por lo que se pretende crear una aplicación más abierta, social y accesible.

1.1. Objetivos

El principal objetivo del proyecto es aplicar técnicas de metodologías de desarrollo ágil, cada día más utilizadas con buenos resultados, en contraposición a los esquemas de trabajo tradicionales del desarrollo en cascada. Dicho desarrollo se realizará siempre con la intención de crear un producto final funcional en un entorno de producción, para poder ser utilizado por usuarios reales. Tener un producto en producción requiere normalmente de cierto gasto económico (dominio web, servidores para la publicación de la aplicación, herramientas de terceros...). Dado que sólo una persona se encarga de todo el proceso, se intentará siempre minimizar los costes, manteniendo la mejor calidad posible en la aplicación. Para conseguirlo se usarán, en la medida de lo posible, herramientas y recursos de código abierto y se llevará el desarrollo de una forma pública y abierta.

Por ello, los principales objetivos del proyecto son:

1. Aplicación de técnicas de metodologías de desarrollo ágil durante el desarrollo de la aplicación.
2. Desarrollo de la aplicación en pasos o iteraciones para llegar a un estado en el que el juego cubra las principales características de una liga fantástica.
3. Preparación de un entorno de producción accesible desde la Web que pueda ser utilizado por usuarios externos al proyecto.
4. Utilización de herramientas y recursos de código abierto para minimizar los costes de desarrollo y mantenimiento de la aplicación en el entorno de producción.

Capítulo 2

Metodología de desarrollo

2.1. Desarrollo Ágil

Uno de los objetivos del proyecto es aplicar técnicas del desarrollo ágil para la resolución del problema. Los primeros enfoques de estas metodologías surgen en la década de los 90, y dan un enfoque revolucionario yendo en contra de los procesos altamente definidos de las metodologías del momento, que se creían necesarias para asegurar plazos, costes y calidad del producto.

Hubo varios planteamientos en su inicio, pero lo fundamental consistía en enfocar los proyectos para pequeños grupos de programadores en entornos altamente productivos. No es hasta 1996 cuando la compañia de automóviles *Chrysler* pretende renovar su sistema de compensación para empleados, un sistema de mala calidad y que tenía que manejar datos de 10.000 empleados. Kent Beck decide comenzar el proyecto de nuevo siguiendo una nueva metodología propia llamada **Programación Extrema** (más conocida en inglés como *Extreme Programming* o *XP* y descrita en [1]).

Las principales características de esta metodología son:

- **Pequeñas publicaciones:** Empezar desarrollando pequeñas características, que sean funcionales por sí mismas. Hacer publicaciones de la aplicación a menudo con mejoras y nuevas funcionalidades.
- **Mantener el diseño simple:** Mantener el diseño lo más simple posible. Los requisitos siempre pueden cambiar y es mejor tener funcional lo más estrictamente necesario.
- **Pruebas unitarias continuas:** Ante una nueva funcionalidad, plantear primero las pruebas, y crear después el código que las satisface. Cargar las pruebas lo más a menudo posible.
- **Programación en parejas:** (Más conocido en inglés por *Pair Programming*) Se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. El código es revisado y discutido mientras se escribe, considerando que la mejora en la calidad es más importante que la posible pérdida de productividad inmediata.
- **Refactorización:** Reescribir ciertas partes del código que sean complejas o estén duplicadas para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.
- **Propiedad del código compartida:** Ningún programador debe ser propietario de una parte de la aplicación. Cualquiera debe ser capaz de trabajar en cualquier parte de la aplicación.

- **Integración Continua:** Todos los cambios deben ser integrados en la aplicación principal al menos una vez al día, y la batería de pruebas validada antes y después de la integración.
- **Seguir estándares de codificación:** Todo el equipo debe seguir los mismos estándares a la hora de escribir código. Idealmente no debería poder distinguirse quién ha escrito qué mirando una parte del código de la aplicación.

Aunque el proyecto no llegó a finalizarse, la *Programación Extrema* se afianzó como metodología y sigue refinándose. Es finalmente en 2001 cuando 17 importantes figuras del desarrollo de aplicaciones se reúnen y redactan lo que se conoce como el *Manifiesto para el Desarrollo Ágil* [3], donde se definen los principios del desarrollo ágil actual. Varios de los autores formaron además la *Agile Alliance*[1], una organización sin ánimo de lucro cuyo principal objetivo es promover los conceptos del desarrollo ágil.

Los principales valores del desarrollo ágil según dicho manifiesto son:

- **El individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas:** La gente es el principal factor de éxito de un proyecto de desarrollo. Es más importante construir un buen equipo que construir el entorno. Muchas veces se comete el error de construir primero el entorno y esperar que el equipo se adapte automáticamente. Es mejor crear el equipo y que éste configure su propio entorno de desarrollo en base a sus necesidades.
- **Desarrollar aplicaciones que funcionen, más que conseguir una buena documentación:** La regla a seguir es no producir documentos a menos que sean necesarios de forma inmediata para tomar una decisión importante. Estos documentos deben ser cortos y centrarse en lo fundamental.
- **La colaboración con el cliente frente a la negociación de un contrato:** Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito.
- **Responder a los cambios en vez que seguir estrictamente un plan:** La habilidad de responder a los cambios que puedan surgir a lo largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, etc.) determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta, sino flexible y abierta.

El objetivo principal de las metodologías ágiles es permitir a los equipos desarrollar aplicaciones rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto. Éstas ofrece una alternativa a los procesos de desarrollo de aplicaciones tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas.

[1] Agile Alliance: `http://www.agilealliance.org/`.

Algunas diferencias entre las metodologías ágiles y las tradicionales pueden verse en la tabla 2.1.1:

Metodologías ágiles	Metodologías tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparadas para cambios durante el proyecto	Mayor dificultad para adaptarse a nuevos requisitos surgidos en fases avanzadas de desarrollo
Consensuadas internamente (por el equipo de desarrollo)	Especificadas ajenamente al proyecto
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (menos de 10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos en diferentes ubicaciones
Pocos roles	Mayor número roles

Cuadro 2.1.1: Diferencias entre metodologías ágiles y tradicionales.

Por supuesto, las metodologías ágiles también tienen sus detractores. Una de las principales críticas hacia ellas es que están más centradas en el desarrollador que en el usuario, ya que el enfoque de los procesos hacia el desarrollo de requerimientos puede afectar al diseño del producto final. Pueden también ser ineficientes para grandes empresas y determinados proyectos.

Otros desarrolladores se acercan más a la idea de que, dependiendo del proyecto, pueden coexistir, mezclando elementos de ambas metodologías (*CMMI or Agile: Why Not Embrace Both*[10]).

Existen otras muchas metodologías ágiles, como *Scrum* o *Crystal Clear*, pero en este proyecto nos centramos básicamente en las aquí expuestas, la *Programación Extrema* y el *Manifiesto Ágil*, ya que son las que mejor se adaptan al modelo de desarrollo que se desea seguir.

2.2. Desarrollo dirigido por pruebas

Uno de los aspectos más importantes de una metodología ágil son las pruebas. Algunas de las premisas planteadas en el apartado anterior se sostienen sobre ellas, por eso es muy recomendable seguir unos procesos de programación orientados a pruebas durante el desarrollo de una aplicación.

Para ello, en este proyecto aplicamos principios del desarrollo dirigido por pruebas (más conocido en inglés como *TDD* o *Test-driven development*), también considerados como iniciados por Kent Beck [2], durante el proyecto en *Chrysler*. Su principal directriz es escribir las pruebas antes que el código.

La secuencia de pasos puede verse gráficamente en la figura 2.2.1 y es descrita en el libro de Kent Beck [2] de la siguiente manera:

1. **Escribir una prueba:** El primer paso es elegir un requerimiento y escribir una prueba para el mismo. La principal diferencia con la escritura de las pruebas después de la implementación es que el desarrollador ya ha pensado en el requerimiento antes de escribir el código.
2. **Ejecutar todas las pruebas y ver que falla la nueva:** La prueba escrita en el primer punto fallará, ya que no existe una implementación para el requerimiento. De esta forma comprobamos que el resto de pruebas siguen cumpliéndose, y en el caso de que la nueva prueba también se cumpla, seguramente no la estemos planteando bien y deberíamos volver al paso anterior.
3. **Escribir el código:** Ahora debe escribirse el código necesario para que el requerimiento pase la prueba. Puede no ser un código muy refinado, pero tiene que centrarse únicamente en resolver el requerimiento, nunca intentando llegar más allá.
4. **Ejecutar todas las pruebas:** Mientras no se validen todas las pruebas deberemos seguir escribiendo código. En el momento que todas las pruebas se cumplan, podremos pasar a los pasos finales con la confianza de que el requisito funciona como queremos.
5. **Refactorizar el código:** En este punto podemos mejorar el código y eliminar elementos duplicados. Deberemos pasar continuamente las pruebas, para asegurarnos de que la funcionalidad sigue intacta.
6. **Repetir:** Finalmente debemos repetir el proceso para cubrir todos los requerimientos. Es recomendable hacer estos pasos lo más pequeños posibles, cubriendo siempre funcionalidades lo más pequeñas posibles.

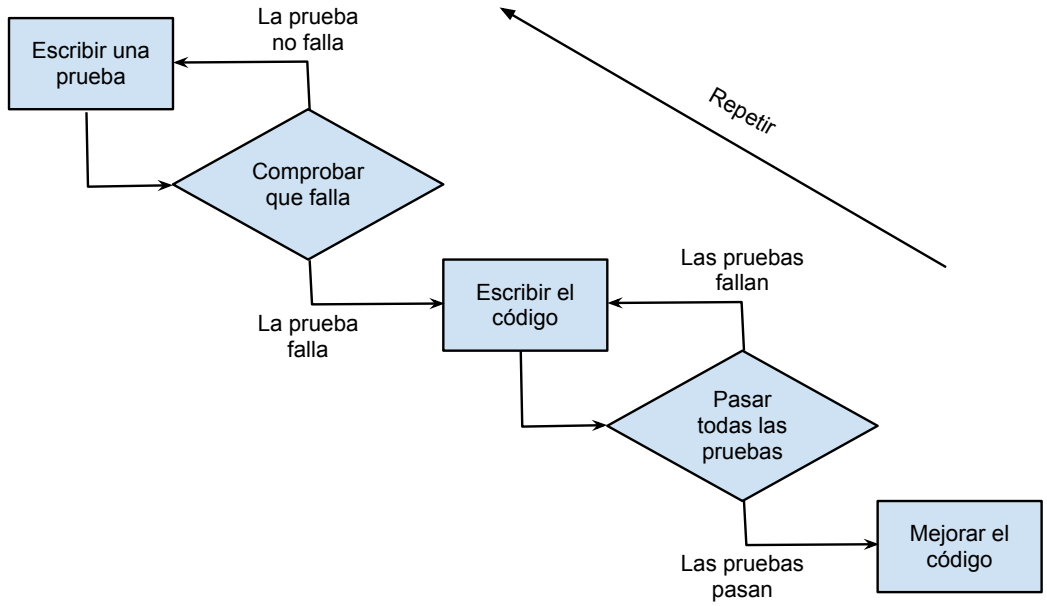

Figura 2.2.1: Proceso TDD.

Una de las ventajas que aporta seguir este tipo de procedimientos es que para proyectos nuevos en los que los apliquemos, en raras ocasiones tendremos la necesidad de depurar el código. A pesar de los elevados requisitos iniciales de aplicar esta metodología, el desarrollo dirigido por pruebas puede proporcionar un gran valor añadido a la aplicación, produciendo aplicaciones de más calidad y en menos tiempo. Este método ofrece más que una simple validación del cumplimiento de los requisitos, pudiendo también guiar el diseño de un programa. Centrándose en primer lugar en los casos de prueba uno debe imaginarse cómo los clientes utilizarán la funcionalidad (en este caso, los casos de prueba). El poder del TDD radica en la capacidad de avanzar en pequeños pasos cuando se necesita. Permite que un programador se centre en la tarea actual y la primera meta es, a menudo, hacer que la prueba pase. Otra ventaja es que, cuando es utilizada correctamente, se asegura de que todo el código escrito está cubierto por una prueba, lo que da al programador un mayor nivel de confianza en el código.

Para ayudar a todo este proceso, es común usar herramientas para la ejecución automática de pruebas. En el caso de este proyecto se utiliza *Rspec*[2], un entorno de pruebas que permite la escritura de las mismas de una forma más descriptiva y flexible que otros. Otra herramienta utilizada es *Guard*[3], que nos permite ejecutar las pruebas de forma automática cada vez que hacemos cambios en el código. Monitoriza los ficheros del proyecto y lanza las pruebas adecuadas cuando detecta alguna modificación. De esta forma siempre nos aseguramos de estar ejecutando las pruebas del código con el que estamos trabajando, ayudando a principios de las metodologías ágiles como la **integración continua**. Éstas y otras recomendaciones han sido obtenidas del libro *Everyday Rails Testing with RSpec* [14].

[2]Rspec: `http://rspec.info/`.
[3]Guard: `https://github.com/guard/guard`.

Capítulo 3

Tecnologías empleadas

3.1. Lenguaje y entorno: *Ruby on Rails*[1]

Ruby on Rails es un entorno de desarrollo de aplicaciones web de código abierto, creado en Julio de 2004 por David Heinemeier Hansson. Es un lenguaje relativamente joven, pero sus virtudes y la gran comunidad que tiene detrás le han hecho ganar mucha importancia en el mundo del desarrollo web.

El entorno está montado sobre el lenguaje de programación *Ruby*[2], creado por el programador japonés Yukihiro *Matz* Matsumoto en 1995, que combina una sintaxis inspirada en los lenguajes de programación *Python*[3] y *Perl*[4] con características de programación orientada a objetos similares a *Smalltalk*[5]. Es un lenguaje de programación interpretado, orientado a objetos, reflexivo y soporta meta-programación, lo que permite que tenga una sintaxis muy legible y elegante. Dispone además de *RubyGems*[6], una herramienta para la distribución de bibliotecas y aplicaciones (*Rails* es una de ellas), donde podemos encontrar gran cantidad de utilidades desarrolladas y mantenidas por la comunidad para ayudarnos en nuestro trabajo.

Sus principales características son:

Modelo-Vista-Controlador

El patrón Modelo-Vista-Controlador [4] fue formulado en 1979 por Trygve Reenskaug como parte de un sistema desarrollado en *SmallTalk* mucho antes que la web y los ordenadores personales. Sin embargo, es el principal paradigma de los entornos de desarrollo web actuales. Separa en tres partes claramente diferenciadas la estructura de la aplicación: el contenido de los datos (modelo), los procesos sobre ellos (controlador) y la presentación de los mismos (vista).

[1] *Ruby on Rails*: `http://rubyonrails.org/`.
[2] *Ruby*: `http://www.ruby-lang.org/`.
[3] *Python*: `http://www.python.org/`.
[4] *Perl*: `http://www.perl.org/`.
[5] *Smalltalk*: `http://www.smalltalk.org/`.
[6] *RubyGems*: `http://rubygems.org/`.

Esta organización permite una mejor abstracción e integración entre ellas, lo que da una gran ventaja a las aplicaciones que encajen con el patrón. Su aplicación al paradigma web puede verse claramente con la habitual estructura de base de datos (modelo), acciones *CRUD*[7] sobre los datos (controlador) y la página web que finalmente llega al navegador del usuario (vista).

Active Record
Active Record [9] es un patrón de diseño declarado por Martin Fowler en 2003, que define una serie de interfaces, propiedades y métodos con los que se debe acceder a un modelo de datos. De esta forma, a nivel de aplicación, podemos abstraernos tanto del sistema de bases de datos a utilizar como del lenguaje del mismo. Con el modelo *Active Record* de *Ruby on Rails*, tenemos mapeados los datos contenidos en nuestra base de datos en objetos lógicos sobre los que realizar acciones, como actualizar o borrar, sin necesidad de escribir sentencias *SQL*[8].

Convención sobre configuración
En lugar de obligar al desarrollador a configurar el entorno por completo, este paradigma busca reducir el número de decisiones, ganando en simplicidad sin perder flexibilidad. Por ejemplo, si en nuestra aplicación tenemos *equipos* y *jugadores* y necesitamos relacionarlos, si la convención dice que la relación entre elementos ha de representarse con la unión de los nombres por orden alfabético, creando el elemento *equipos_jugadores* ya tendremos la relación establecida sin necesidad de configurarla en ningún sitio.

No te repitas
Más conocido en inglés como *Don't Repeat Yourself* o *DRY*, este principio de programación (citado por primera vez en *The Pragmatic Programmer* [11]) defiende la reducción de repetición de información, haciendo que cada porción de información sea única y reutilizable dentro de una aplicación, mejorando la claridad y evitando posibles inconsistencias a medida que surjan cambios. Un ejemplo claro en el proyecto son los equipos de los usuarios y los equipos reales, que comparten gran parte de su lógica, la cual está referenciada en un único sitio.

Estas características, la versatilidad del lenguaje Ruby y la experiencia ya adquirida en el entorno son las principales razones para elegir a *Ruby on Rails* frente a otros entornos web también en alza, como *Django*[9] (*Python*) o *NodeJS*[10] (*JavaScript*).

Ruby on Rails cuenta además con unas completas guías de referencia [7] disponibles en la web.

3.2. Gestión de código: *Git*[11] / *GitHub*[12]

Cualquier proyecto de desarrollo de aplicaciones actual debe gestionarse con un sistema de control de versiones con el que mantener un registro de todos los cambios realizados en la aplicación. Esto nos permite poder recuperar el estado de nuestro proyecto en cualquier punto de su desarrollo. Ésta es suficiente razón para usar uno en este proyecto, pero para otros en los que colaboren varios programadores, facilita el trabajo concurrente y ofrece herramientas para solucionar los diferentes conflictos que puedan surgir cuando dos personas trabajan simultáneamente sobre una misma cosa.

[7] *CRUD*: Siglas en inglés de creación, lectura, actualización y borrado
[8] *SQL*: lenguaje declarativo de acceso a bases de datos.
[9] *Django*: `https://www.djangoproject.com/`.
[10] *NodeJS*: `http://nodejs.org/`.
[11] *Git*: `http://git-scm.com/`.
[12] *GitHub*: `http://www.github.com`.

El sistema usado para el proyecto es *Git*, un sistema de control de versiones diseñado por Linus Torvalds en 2005, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Su gran diseño lo deja muy por encima de otras alternativas algo más antiguas pero muy extendidas como *Subversion*[13].

Existen otras alternativas más actuales y eficientes como *Mercurial*[14], pero la gran difusión de *Git* entre la comunidad de desarrolladores de código abierto y la gran cantidad de proyectos que se siguen migrando a este sistema lo siguen manteniendo por delante de los demás.

Hay muchos recursos sobre el uso *Git* disponibles en la red. Destacan especialmente los libros *Pro Git* [6] y *Git Internals* [5], por ser de los más completos y extendidos, y que han sido de gran utilidad durante el desarrollo de este proyecto.

Como plataforma para alojar el repositorio de código de *Git* se ha optado por *GitHub*. Es una herramienta web para alojar proyectos que usan *Git* como control de versiones. Está desarrollada originalmente con *Ruby on Rails*. Ofrece servicio gratuito para proyectos públicos, además de herramientas colaborativas y de gestión de proyectos como por ejemplo un sistema de hitos para marcar las diferentes fases del proyecto.

Aunque existen otras plataformas con las mismas características como *BitBucket*[15] (que incluso ofrece repositorios privados gratuitamente), *GitHub* es una de las compañías más importantes en el mundo *Rails*, con una gran comunidad y cuyos servicios están integrados con múltiples herramientas que se verán en los siguientes puntos de esta documentación.

3.3. Base de datos: *PostgreSQL*[16]

Para implementar el modelo de datos sobre el que funcionará la aplicación, se hace uso de una base de datos relacional. Dado que uno de los objetivos es reducir gastos manteniendo la calidad, los dos sistemas de bases de datos no propietarios más utilizados en la actualizad son *MySQL*[17] y *PostgreSQL*.

Aunque *MySQL* es un sistema más extendido que ofrece muy buena velocidad y rendimiento, *PostgreSQL* ofrece mejor escalabilidad, algo muy necesario en el mundo web actual donde un aumento inesperado de usuarios puede bloquear un sistema si no está preparado. Además implementa *rollbacks*[18] (*MySQL* no lo había incluido hasta las últimas versiones), subconsultas e integridad transaccional[19], la cual es necesaria en algunos puntos del proyecto.

[13] *SVN*: `http://subversion.apache.org/`.
[14] *Mercurial*: `http://mercurial.selenic.com/`.
[15] *BitBucket*: `https://bitbucket.org/`.
[16] *PostgreSQL*: `http://www.postgresql.org/`.
[17] *MySQL*: `http://www.mysql.com/`.
[18] Capacidad para volver al estado inicial al interrumpirse una acción.
[19] Ejecutar una serie de acciones de forma atómica (o todas o ninguna).

Por estas razones y que como veremos, el sistema de despliegue inicialmente elegido (*Heroku*) requiere el uso de *PostgreSQL*, éste último es el elegido para la aplicación.

3.4. Despliegue y alojamiento: *Heroku*[20] / *Openshift*[21]

Para el despliegue de la aplicación desde el entorno de desarrollo local hasta un entorno de producción publicado en la web, se requiere de un servidor web configurado y preparado con el entorno para poder alojarla. Además, ha de hacerse un mantenimiento del mismo cada vez que actualizamos la aplicación y añadimos funcionalidades que hagan uso de alguna nueva biblioteca.

Heroku es una plataforma de publicación y alojamiento de aplicaciones en la nube desarrollada con *Ruby on Rails*, que facilita y automatiza estas tareas. Soporta gran variedad de lenguajes de programación. Permite desplegar una aplicación utilizando su repositorio de código (*Git*), recoge la rama indicada del repositorio, crea una instancia de una máquina virtual usando los servicios de *Amazon EC2*[22] para computación en la nube, instala y configura todo lo necesario para el funcionamiento de la aplicación y, finalmente, la publica en la web sin necesidad de mantenimiento.

Puedes publicar tantas aplicaciones como quieras gratuitamente. Cada aplicación dispone de 5 megas de base de datos *PostgreSQL* y una sola máquina virtual. Estas dos características pueden ser ampliadas previo pago, convirtiéndola en una plataforma ideal para comenzar un desarrollo de bajo coste y escalar progresivamente el servidor según vaya necesitando más recursos.

Uno de los puntos negativos de la plataforma es que no ofrece espacio de almacenamiento estático (la máquina virtual se regenera con cada despliegue), de modo que todos los ficheros que no estén en el repositorio de código se pierden, como por ejemplo las imágenes y contenidos creados desde la propia aplicación. Esta carencia es subsanada usando el servicio de almacenamiento de ficheros en la nube de *Amazon S3*[23], que también ofrece unos servicios mínimos gratuitos y la posibilidad de aumentar el espacio pagando según se vaya necesitando.

Heroku fue la primera opción elegida durante el proceso de desarrollo, por su integración con *Ruby on Rails* y por su gran comodidad, pudiendo hacer el despliegue de toda la aplicación con un sólo comando. Pero con la cantidad de datos almacenada cada semana, en unos meses se alcanzó el límite gratuito de base de datos (diez mil filas). Llegados a este punto habría que pagar 9 dólares al mes por una base de datos de más capacidad, por lo que se encontró una alternativa que no se había descubierto al comienzo del proyecto. *Openshift* es una plataforma similar a *Heroku* (funciona de forma parecida a la descrita), pero desarrollada por *RedHat*[24], una de las compañías que más apoya el código libre y que mantiene una distribución de *Linux* con el mismo nombre. Las diferencias son que ofrece hasta 3 aplicaciones gratuitas pero con un giga de almacenamiento de datos, por lo que no tenemos una limitación tan reducida para la base de datos. Además, permite más versatilidad a la hora

[20] *Heroku*: `http://heroku.com`.
[21] *OpenShift*: `http://openshift.redhat.com`.
[22] *Amazon EC2*: `http://aws.amazon.com/es/ec2/`.
[23] *Amazon S3*: `http://aws.amazon.com/es/s3/`.
[24] *RedHat*: `http://www.redhat.com/`.

de configurar la máquina, pudiendo conectarse a ella directamente pero manteniendo la posibilidad de controlarlo todo desde el proceso de despliegue sin mantenimiento.

Por esto, finalmente se migró toda la información de una plataforma a otra, y *Openshift* es la actual plataforma de despliegue.

3.5. Integración continua: *Travis*[25]

La integración continua fue propuesta por primera vez como parte de la Programación Extrema, que propone que varias veces al día todas las ramas de desarrollo de una aplicación se integren y se prueben, para evitar los grandes problemas que puedan surgir en procesos de desarrollo largos.

Para automatizar este proceso lo que se hace en la práctica es que cada vez que se añade nuevo código a la rama principal de la aplicación toda ella es montada desde cero, y se lanza la batería de pruebas (por supuesto, esto tiene sentido usando una metodología de desarrollo dirigida por pruebas). De esta forma, tendremos cierta seguridad de que nuestra aplicación no tenga errores graves y nunca pondremos en producción una aplicación inestable.

Existen varios sistemas libres de integración continua, como *Hudson*[26] o *Jenkins*[27], que soportan múltiples lenguajes. Uno de los más acogidos recientemente en la comunidad *Rails* es *Travis*, cuya principal ventaja y razón para ser el elegido en este proyecto, es su integración con *GitHub* (el sistema de gestión de código utilizado) y el alojamiento gratuito para aplicaciones libres albergadas en ese mismo repositorio.

Cada vez que actualizamos el código de la aplicación *Travis* lo descarga automáticamente, inicia una máquina virtual, instala y configura la aplicación, lanza la batería de pruebas y, en el caso de encontrar algún problema, nos envía un aviso para que lo solucionemos.

[25] *Travis*: `https://travis-ci.org/`.
[26] *Hudson*: `http://hudson-ci.org/`.
[27] *Jenkins*: `http://jenkins-ci.org/`.

3.6. Mantenimiento de código

Otro punto importante durante el desarrollo y mantenimiento de una aplicación es controlar la calidad del código. Éstas son tres herramientas integradas en el proyecto para ayudar en este aspecto:

3.6.1. *Gemnasium*[28]

Gemnasium es un servicio web para aplicaciones *Ruby on Ralis* que monitoriza un repositorio de código y comprueba que las dependencias de la aplicación están al día. Se integra con *GitHub*, con lo que es capaz de saber cuándo se actualiza el código y de comparar las versiones de las bibliotecas usadas en la aplicación con las versiones actuales de las mismas. Todas las semanas envía un resumen de las nuevas versiones que pueden ser actualizadas indicando, si es posible, cuáles son los cambios en ellas. Además, ofrece un plan gratuito para todas las aplicaciones públicas alojadas en *GitHub*.

3.6.2. *Code Climate*[29]

Code Climate es un servicio web para aplicaciones *Ruby* que monitoriza un repositorio de código y comprueba la calidad del contenido del mismo, buscando partes en las que se aplican malas prácticas, como repetición de código (*DRY*, citado anteriormente), funciones demasiado complejas que pueden ser *refactorizadas*[30], ficheros demasiado grandes... La aplicación puntúa cada fichero con una nota y ofrece consejos sobre cómo mejorar la calidad. Como otros servicios vistos hasta ahora, se integra con *GitHub* y ofrece un plan gratuito para todas las aplicaciones públicas alojadas en el servicio.

3.6.3. *Errbit*[31]

Errbit es una aplicación escrita en *Ruby on Rails* para detectar los errores que se producen en una aplicación en producción. Cuando una aplicación está siendo usada por un usuario y se produce un error inesperado, el programador no tiene por qué tener constancia del problema si el usuario no da parte. Con esta aplicación, los errores son almacenados en un sistema separado, junto con toda la información posible de la causa del error, y se envía una notificación por correo al equipo de desarrollo para que lo revise lo antes posible.

En este caso, *Errbit* no es un servicio externo al igual que las anteriores herramientas, por lo que hay que instalarlo y mantenerlo en un servidor propio. Hemos aprovechado la facilidad de despliegue de *Heroku* para alojar allí la aplicación *Errbit* para el proyecto.

[28] *Gemnasium*: `https://gemnasium.com/`.
[29] *Code Climate*: `https://codeclimate.com/`.
[30] Dividir una funcionalidad en otras más pequeñas sin modificar su comportamiento.
[31] *Errbit*: `https://github.com/errbit/errbit`.

Capítulo 4

El juego

4.1. Descripción

onLeague es un juego vía web de simulación sobre la gestión de un equipo de fútbol profesional. Los usuarios deben registrar una cuenta en la aplicación para poder crear un equipo. Una vez creado, pueden fichar jugadores de la liga en la que estén jugando, cumpliendo las distintas restricciones de presupuesto y jugadores. Disponen además de 3 cambios semanales una vez hayan creado dicho equipo. En función de la actuación de los jugadores en los partidos reales, los equipos van consiguiendo puntos. El objetivo del usuario es conseguir más puntos que el resto de equipos, tanto semanalmente como en la clasificación de la temporada.

La aplicación está preparada para manejar múltiples ligas simultáneamente, centrando siempre los equipos y clasificaciones en una sola. Debido a la gran cantidad de datos a gestionar, actualmente sólo se procesa la liga de fútbol profesional de España o liga BBVA. En el caso de disponer de más recursos para administrar contenidos o publicar la aplicación en otros países, se podrían añadir otras ligas, como la liga de segunda división o la *Premier League* inglesa.

A lo largo de la documentación se utilizan algunos términos o conceptos que pueden considerarse ambiguos en algún punto. Siempre que aparezcan estos términos se deben interpretar según las siguientes definiciones:

- **Jugadores:** jugadores de fútbol reales.
- **Clubs:** clubs reales de fútbol, donde juegan los jugadores.
- **Usuarios:** usuarios de la aplicación.
- **Equipos:** equipos de los usuarios, creados a partir de los jugadores de fútbol reales.

En cuanto a los objetivos a alcanzar con la aplicación, pueden dividirse dos secciones claramente diferenciadas:

1. Interfaz de administración, gestionada por usuarios separados, donde:
    - Gestionar la información de los jugadores.
    - Gestionar los clubs reales, en qué ligas juegan y qué jugadores tienen.

- Gestionar los partidos de las ligas, clubs enfrentados, horarios y eventos que ocurren en los mismos.
- Consultar información del estado de la aplicación, usuarios y datos generales.

2. Interfaz para los usuarios donde pueden:
   - Consultar los partidos de las ligas, tanto los horarios, como los resultados y eventos de los partidos ya jugados.
   - Consultar información de los clubs reales.
   - Consultar información de los jugadores, con detalles sobre sus estadísticas y evolución jornada a jornada.
   - Crear y gestionar sus propios equipos.
   - Consultar las clasificaciones de los equipos por jornada y por temporada completa.

Cada semana los administradores actualizan los datos de los partidos jugados (alineaciones, sustituciones, tarjetas y goles) y automáticamente se calculan los puntos generados por cada jugador.

4.2. Estado del arte

Las ligas fantásticas (conocidas en inglés como *Fantasy Leagues* o *Fantasy Sports*) son juegos en los que los usuarios crean equipos para competir con los de otros usuarios, basándose en un sistema de puntos a partir las estadísticas de los jugadores y equipos de la liga o deporte en cuestión.

Es un tipo de juego muy extendido en otros países. Según informes de la *Fantasy Sports Trade Association*[1] han participado en este tipo de ligas 32 millones de personas a partir de 12 años durante 2010, sólo en Estados Unidos y Canadá, teniendo un impacto económico anual de entre unos 3 y 4 mil millones de dólares. Fuera de estos países no tienen tanta acogida, pero siguen creciendo. En Gran Bretaña por ejemplo, se estimaron unos 6 millones de jugadores y el 80 por ciento de ellos, en ligas de fútbol.

En la actualidad en España no hay una gran oferta de ligas fantásticas. Las más importantes y utilizadas son dos de fútbol: la **Liga Fantástica Marca**[2] y **Comunio**[3].

La Liga Fantástica Marca es una liga fantástica creada por el diario deportivo Marca. En sus inicios funcionaba completamente a través del periódico. Toda la gestión del equipo se realizaba por correo ordinario o por teléfono, y cada semana se publicaban las puntuaciones en su propio diario. Actualmente tiene un sistema web donde gestionar y seguir a tus equipos. Los dos primeros equipos son gratis, pero el resto de acciones como cambios o creación de nuevos equipos hay que pagarlos con dinero real. Por ello, la base del juego es en gran parte monetaria y puede ofrecer premios semanales, lo que atrae a gran cantidad de usuarios. El problema es que limita mucho la jugabilidad para los usuarios que simplemente quieran divertirse. Es decir, es un juego principalmente comercial y orientado al público más fanático del fútbol. En las últimas versiones ha añadido cosas interesantes, como la posibilidad de crear peñas con tus amigos dentro del juego o la de hacer apuestas de dinero con otros usuarios.

Comunio es otra de las ligas fantásticas más jugadas en España. Es también una aplicación web, pero su concepto es algo diferente a la de Marca. En vez de tener una clasificación global, aquí se

[1] Asociación que representa las ligas fantásticas en la industria.
[2] LFM: `https://ligafantastica.marca.es/`.
[3] Comunio: `http://www.comunio.es/`.

forman pequeños grupos privados, con un mercado de jugadores propio en el que los usuarios no pueden elegir jugadores que estén en otros equipos del mismo grupo. Este tipo de liga ofrece un mayor realismo, pero limita las capacidades sociales al estar limitado a grupos pequeños. Se puede jugar gratuitamente y no ofrece premios, pero dispone de dos tipos de usuario de pago, los cuales disponen de más datos y herramientas para gestionar los equipos. Este tipo de modelo de negocio se conoce como *Freemium*[4] y ofrece un acceso gratuito con la posibilidad de contenidos exclusivos mediante pago. Además, estos sistemas no afectan a la jugabilidad de los usuarios gratuitos, y aporta más comodidades que ventajas para los de pago, con lo que no se rompe el balance del juego.

onLeague parte de las ideas básicas de las ligas fantásticas y pretende alcanzar un público más amplio, no limitando al usuario aficionado y facilitando además toda la información posible para no necesitar ser un experto en fútbol, manteniendo una comunidad abierta y global con una clasificación general, pero permitiendo llevar el juego a un nivel más social con amigos.

4.3. Aplicaciones web 2.0

Web 2.0 es el término que define a las aplicaciones web desde que dejaron de ser simples páginas estáticas. Más que una tecnología, puede verse como una actitud, una suma de prácticas que mejoran la experiencia de los usuarios en la web. Las principales características son que el usuario pueda participar en la página, generar contenidos e interacturar con otros usuarios, siempre manteniendo una experiencia de uso rica.

onLeague es un juego y muchas de estas características ya son inherentes. Un usuario no genera contenidos de la forma que puede hacerlo en un blog o en una red social, pero las acciones que realiza en su equipo afectan a una generación dinámica de contenidos, como son las clasificaciones. Se han tenido en cuenta varios aspectos para potenciar las características de la web 2.0:

Integración con redes sociales

La aplicación está integrada con las redes sociales más extendidas por la red, como son *Facebook*, *Twitter* o *Google+*. Estas aplicaciones ofrecen una serie de servicio (mediante el uso del protocolo de seguridad *OAuth*[5]) que permite a sus usuarios interactuar con otras aplicaciones siempre que se den los permisos para ello. En *onLeague* se permite a los nuevos usuarios usar sus cuentas en estas redes sociales, para acceder por primera vez al juego, sin necesidad de darse de alta, algo que facilita mucho el sistema de registro. Además, mientras estén identificados en su red social, siempre pueden acceder a su cuenta de *onLeague* sin necesidad de introducir sus datos.

Esta conexión añade también otro nivel social al juego, ya que los usuarios pueden ver las clasificaciones no sólo de forma global, sino únicamente entre los amigos de sus redes sociales (pueden asociar varias redes a la misma cuenta) que también las tengan asociadas a la aplicación. Esta característica sólo está disponible para *Facebook* y *Twitter*, ya que actualmente *Google* no dispone de un servicio de consulta de contactos.

Interfaz de usuario interactiva

Para ofrecer a los usuarios una experiencia rica y fluida, se aplican técnicas no intrusivas[6] en el lado del cliente[7] mediante el uso de *JavaScript*[8], con librerías *jQuery* y las herramientas de *Twitter Bootstrap*.

[4] Contracción de las palabras inglesas *free* y *premium*.
[5] OAuth 2.0: `http://oauth.net/`.
[6] Separación de la lógica en las vistas para no interferir en la semántica de la web.
[7] Lógica que se procesa en el ordenador del usuario y no en el servidor de la aplicación.
[8] Lenguaje de programación orientado a la funcionalidad en el lado del cliente.

Twitter Bootstrap, desarrollado por trabajadores de *Twitter*, ofrece un conjunto de componentes con plantillas *HTML5*[9] y *CSS3*[10] sobre funcionalidad de *jQuery*. En la aplicación se hace uso de algunas como menú y listas de navegación, botones desplegables, pestañas, mensajes de alerta, ventanas modales[11] para mostrar información sobre los jugadores en cualquier parte de la página, sin tener que perder la navegación al ir a buscarlo en otro sitio.

Además de estos componentes se hace uso de *AJAX*[12] para consultar información del servidor en un segundo plano y que el usuario no pierda el foco de lo que está haciendo. Por ejemplo, las ventanas modales con información de los jugadores no se cargan inicialmente con la página, ya que en páginas con muchos jugadores, el tamaño de la información cargada al inicio aumentaría considerablemente. Cuando un usuario selecciona un jugador se solicita sólo esa información al servidor mediante *AJAX*, y se carga en la ventana emergente, sin que el usuario note diferencia. Otro claro ejemplo lo encontramos en el editor de equipos, en el que tanto la búsqueda de jugadores, como la compra y venta de los mismos se realiza con llamadas *AJAX* al servidor, recargando las secciones concretas que se ven afectadas por las acciones y evitando que los usuarios tengan que navegar por distintas secciones de la página.

Tanto *jQuery* [8] como *Twitter Bootstrap* [15] disponen de completas guías de consulta mantenidas y actualizadas.

Internacionalización

La aplicación está preparada para visualizarse tanto en español como en inglés. Inicialmente sólo pretende acercarse al público español, pero se ha implantado la internacionalización porque es un proceso difícil de introducir si no se ha llevado correctamente desde el principio. Este aspecto está completamente separado de la funcionalidad y vistas de la aplicación, para que la traducción o adicción de más idiomas sea sencilla.

Están internacionalizados tanto los textos estáticos como las direcciones URL[13] de las páginas y los contenidos dinámicos en base de datos. Estos contenidos pueden introducirse desde la interfaz administración en los diferentes idiomas definidos en la aplicación.

Direcciones *URL* amigables

Otro aspecto que se ha tenido en cuenta es la generación de unas direcciones URL para los distintos recursos de la aplicación lo más legibles posibles, para que un usuario pueda recordarlas fácilmente o identificarlas sólo con verlas. Por ejemplo, el recurso para acceder a la página de un equipo viene determinado por el nombre del equipo, la liga en la que juega y la temporada (*/equipos/nombre-liga-bbva-2013*). O para consultar los partidos que vienen clasificados por temporada y jornada, en vez de usar los clásicos parámetros para seleccionarlas, se introducen directamente como parte del recurso (*/partidos/2013/1*).

Estas prácticas ayudan, además, a optimizar el posicionamiento de la página en motores de búsqueda (más conocidas es inglés como prácticas *SEO* o *Search Engine Optimization*).

Metadatos

Para ayudar también a estas prácticas *SEO*, las distintas páginas públicas que pueden ser accedidas por buscadores, ofrecen metadatos[14] dinámicos en función de los contenidos que ofrecen. Estos metadatos son utilizados por los buscadores para conocer más información del contenido que ofrece una página y dar resultados más precisos en las búsquedas de los usuarios.

[9] Última especificación del lenguaje de marcado de páginas web.

[10] Última especificación del lenguaje de especificación de estilos de páginas web.

[11] Diálogos superpuestos a la aplicación para mostrar información.

[12] JavaScript asíncrono que consulta información del servidor en un segundo plano, para que el usuario no pierda el foco de lo que está haciendo.

[13] Cadenas de textos que definen recursos en Internet.

[14] Datos que describen otros datos.

Por ejemplo, las páginas con los detalles de los partidos incluyen metadatos de título, descripción y palabras clave con la liga, los equipos y los eventos del partido. Usuarios que buscan información sobre un equipo o una liga, podrían llegar a la página a través de una búsqueda.

4.4. Diseño del juego

En esta sección se describe con más detalle el funcionamiento general del juego.

Los usuarios pueden crear dos equipos por cada una de las ligas disponibles (sólo la Liga BBVA actualmente). Una vez un equipo esté activo, comenzará a acumular puntos cada semana en función de las actuaciones de los jugadores reales en los partidos de la liga cada jornada. Con esos puntos consigue entrar en una clasificación con el resto de equipos. El objetivo de los usuarios es obtener más puntos que los demás para quedar primero de la clasificación, por jornada y por temporada.

Para poder activar un equipo, los usuarios disponen de 200 millones para comprar jugadores de fútbol siguiendo las siguientes restricciones:

- Debe tener 11 jugadores.
- Debe tener un portero.
- Debe tener al menos 3 defensas y a lo sumo 5.
- Debe tener al menos 3 centrocampistas y a lo sumo 4.
- Debe tener al menos 1 delantero y a lo sumo 3.
- No puede tener más de 3 jugadores extracomunitarios.
- No puede tener más de 3 jugadores del mismo club.
- Un jugador no puede ser comprado si no se dispone del presupuesto necesario.
- Un jugador no puede ser comprado a partir de una hora antes del partido que juega en la jornada actual.

Estas restricciones están indicadas y controladas durante la creación del equipo, para que el usuario sepa en todo momento qué puede y qué no puede hacer. Mientras un equipo no esté activo no consigue puntos, pero puede comprar y vender jugadores sin ningún tipo de contabilización. Una vez se cumplan todas las restricciones, el equipo puede activarse para empezar a acumular puntos.

A partir de ese momento el usuario sólo puede realizar 3 cambios por jornada, siempre cumpliendo las restricciones anteriores. Se considera un cambio completo la venta de un jugador y la posterior compra de otro. Cuando un usuario compra o vende un jugador, la acción no se hace efectiva directamente, sino que se pasa a un estado de negociaciones donde se puede ver en qué estado se queda el equipo (alineación, presupuesto...). Finalmente, se pueden confirmar las negociaciones, para que los traspasos de los jugadores se realicen, o rechazarlas, con lo que volveremos a tener nuestro equipo inicial. Este proceso hace que los cambios sean flexibles, ya que pueden realizase jugador a jugador o, por ejemplo, pueden pasarse a la venta 3 jugadores para ver de cuanto dinero dispone después de la venta e ir buscando los mejores jugadores con ese presupuesto.

La jugabilidad y diversión del juego radica en las decisiones de los usuarios al determinar la formación y jugadores idóneos de su equipo para la jornada actual. Para ello, los usuarios disponen de toda la información acumulada de los jugadores durante la liga:

- Información personal: nombre, fecha de nacimiento, edad, nacionalidad, club y demarcación.
- Último partido jugado (con su puntuación) y próximo partido.
- Estadísticas de la temporada: puntos totales, goles, goles encajados para porteros, asistencias y tarjetas.
- Estadísticas de partidos: partidos jugados, titularidades, minutos por partido y puntos por partido.
- Popularidad del jugador en el resto de equipos (porcentaje de equipos que poseen al jugador).
- Evolución de puntos por jornada y comparación con la media de jugadores.

Las reglas de puntuación para los jugadores según los distintos eventos son las siguientes (tablas de 4.4.1 a 4.4.4):

Eventos de los goles	**Puntos**
Por un delantero	+2
Por otras demarcaciones	+3
Asistencia de gol	+1
En propia puerta	-2
De penalti	+2
Penalti fallado por fuera	-3
Penalti fallado por parada	-2

Cuadro 4.4.1: Reglas de puntuación para goles.

Eventos de los porteros	**Puntos**
0 goles encajados	+2
1 solo gol encajado	+1
n goles encajados	-n
Penalti parado	+3
Penalti fuera	+1

Cuadro 4.4.2: Reglas de puntuación para los porteros.

Aclaración: cuando un portero encaja un solo gol, gana un punto pero pierde uno por el gol recibido, por lo que se queda con 0 puntos por goles.

Eventos de las tarjetas	**Puntos**
Amarilla	-1
Roja	-1
Roja directa	-3

Cuadro 4.4.3: Reglas de puntuación para las tarjetas.

Aclaración: un jugado sólo puede tener una tarjeta de cada tipo por partido, una tarjeta roja sólo se saca después de una amarilla (es decir que conlleva -2 puntos en total), y una roja directa es una expulsión directa por una falta muy grave. El jugador puede no tener o tener ya una amarilla.

Eventos generales por partido	**Puntos**
Jugar de titular	+2
Entrar por otro jugador	+1
Jugar en el equipo ganador	+1

Cuadro 4.4.4: Reglas de puntuación para las tarjetas.

Los jugadores pueden recibir, además, entre 1 y 3 puntos según una valoración objetiva de su actuación en el partido. De esta forma, un jugador que no haya conseguido puntos por los eventos del partido pero haya tenido una buena actuación obtiene una puntuación más justa.

4.5. Contenidos

Otra parte importante del sistema son los datos a introducir. Para que el juego funcione y los jugadores obtengan puntos, la aplicación necesita ser alimentada con los datos reales de las ligas. Por una parte, con los clubs y los jugadores que se mantienen toda la temporada y, por otra, con los partidos jugados cada semana.

La mejor opción para cubrir esta necesidad, es encontrar una fuente de datos fiable, con servicios que los hagan accesibles desde Internet para poder ser consultados automáticamente por nuestra aplicación. Existen servicios con estas características en el mundo de las apuestas deportivas. La información va orientada a las apuestas, pero suelen ofrecer estadísticas, donde podríamos obtener la información necesaria. El problema es que estos servicios son de pago, con un alto coste y normalmente usados por grandes empresas de apuestas. *BetRadar*[15] es una de las más importantes, en la que un paquete básico de servicios está entre los 1.000 y 5.000 euros al mes. Se encontraron otras opciones de no tan alto nivel, pero todas con grandes costes.

De todos los servicios encontrados, lo ideal sería utilizar *Openfooty*[16]. Es una aplicación web que ofrece datos deportivos (no orientados a apuestas) de forma gratuita mediante una *API*[17] *JSON*[18]. Sus servicios ofrecen los datos necesarios para nuestras necesidades. Al ser gratuita, el acceso es

[15] BetRadar: `https://www.betradar.com`.
[16] Openfooty `http://www.footytube.com/openfooty/`.
[17] Interfaz de programación de aplicaciones, "bibliotecas"de consulta entre aplicaciones.
[18] *JavaScript Object Notation*, formato ligero para el intercambio de datos.

limitado, y aún habiéndolo solicitado y estar en lista de espera, tras varios meses todavía no ha sido concedido.

Otra posibilidad a probar para extraer información es la Web semántica. Son un conjunto de técnicas desarrolladas por *World Wide Web Consortium*[19] para publicar datos legibles por aplicaciones informáticas añadiendo metadatos semánticos y ontológicos. Estos metadatos se proporcionan de manera formal para que puedan ser evaluados por las aplicaciones. La *DBpedia*[20] es un proyecto para extraer datos de la *Wikipedia*[21] y ofrecerlos en forma de Web semántica, pero tras probar varias consultas los datos obtenidos eran insuficientes y desactualizados.

Finalmente, la única opción viable es hacerlo de forma manual. Para el caso de los datos de las ligas es relativamente asequible. Durante el verano y antes de dar comienzo la liga, la información puede ser consultada de varias fuentes, como las propias páginas de los clubs, la *Wikipedia* o diferentes revistas[12] que los medios especializados publican con información recopilada sobre todos los clubs de la liga. Una vez creados los datos en el sistema, sólo se necesita ir actualizando los clubs, moviendo los jugadores de unos a otros y añadiendo los nuevos durante las dos épocas del año en las que los clubs pueden hacerlo, el mercado de verano y el mercado de invierno.

Los datos para los partidos son más problemáticos. Con una sola liga tenemos cada semana datos sobre 10 partidos, los cuales tienen 22 jugadores titulares, 6 cambios, 28 puntuaciones, unos 3 goles y unas 6 tarjetas de media. Esto hace unas 650 instancias de eventos por semana. Para añadir todos estos datos manualmente se pueden consultar los diferentes medios deportivos digitales. Aunque son bastante fiables, es recomendable usar varias fuentes para contrastar porque puede haber errores.

Esta tarea conlleva mucho más tiempo y esfuerzo que la anterior descrita, y requiere cierto nivel de consistencia para para minimizar la posibilidad de cometer errores. Por esto, para ayudar se ha aplicado una técnica de *Web scraping*[22], una técnica de extracción de datos desde la web. Mediante una aplicación se simula el acceso humano a una página web y se procesa el código de la página obtenida para recoger los datos deseados. Esto tiene varios problemas. Al ser dependiente del código de la página, cualquier cambio en ésta puede ocasionar el mal funcionamiento del proceso. Con una *API*, estos cambios serían notificados a los desarrolladores para poder realizar las adaptaciones necesarias, en cambio con *Web Scraping* lo más posible es no darse cuenta hasta que el error en el proceso se produce. El otro punto sensible con esta técnica son los temas legales respecto a la página que se consulta. En nuestro caso, los datos a los que se acceden son públicos y no se rompe con los términos de servicio de las páginas.

En cualquier caso, el *Scraping* en el proyecto es utilizado como una herramienta manual para la ayuda de carga de los datos. No se utiliza nunca de forma automática y masiva para no afectar a los sistemas de las páginas externas. Además, los datos obtenidos nunca son creados directamente en el sistema, únicamente se inicializan para que una persona los revise y contraste con otras fuentes porque, como se indicaba antes, pueden existir errores.

[19] Consorcio internacional que produce recomendaciones para la Web.
[20] DBpedia: `http://es.dbpedia.org/`.
[21] Wikipedia: `http://www.wikipedia.org`.
[22] Recolección web.

Capítulo 5

Desarrollo ágil

Las principales funcionalidades de la aplicación están claramente separadas en dos grupos de requisitos: los de la aplicación de administración y los de la aplicación de usuario. Veremos primero a grandes rasgos cuáles son y, posteriormente serán desglosados en las distintas iteraciones llevadas a cabo por el desarrollo ágil.

Interfaz de administración:

- Creación y gestión de ligas, clubs y jugadores.
- Creación y gestión de partidos.
- Creación y gestión de eventos de partidos: alineaciones, sustituciones, tarjetas y goles.
- Cálculo automático de puntos.
- Consulta de información del estado de la aplicación, usuarios y datos generales.

Interfaz de usuario:

- Creación, configuración de usuario y vinculación con redes sociales.
- Consulta de información de la liga (clubs y partidos).
- Creación y gestión de equipos.
- Consulta de información de los jugadores, con detalles sobre sus estadísticas y evolución jornada a jornada.
- Consulta de las clasificaciones de los equipos por jornada y por temporada, e integración con redes sociales.

5.1. Consideraciones previas

Como se ha descrito anteriormente, para el desarrollo del proyecto se aplican distintos principios del desarrollo ágil siguiendo una serie de iteraciones para implementar funcionalidades que tienen sentido en sí mismas y que se irán completando en las sucesivas iteraciones. Éstas serán descritas con más detalle a continuación y siguen la siguiente estructura:

- Diagramas piramidales para describir el estado del sistema y cómo se van integrando las distintas secciones.
- Descripción detallada de los requisitos principales de la iteración.
- Relaciones y descripción de los nuevos modelos implicados en la iteración. Se detallan los atributos más relevantes. Todos los modelos tienen un identificador numérico único llamado *id* en la base de datos, usado para la identificación y relación entre modelos, que es obviado en las descripciones.
- Breve descripción de los elementos de las vistas afectadas por la iteración y diagrama de navegación de las nuevas secciones de la web. Se dividen los diagramas de navegación en la interfaz de usuario (con cajas azules para las secciones públicas y rojas para las privadas) y en la interfaz de administración (con cajas verdes y siempre de acceso privado). Lo ideal en este punto es contar además con la participación de un diseñador para realizar un boceto visual previo de cada vista. Con la limitación de recursos se opta por describir las vistas y maquetarlas durante la implementación.
- Especificación de las pruebas previas al desarrollo que determinan la programación. Las pruebas intentan siempre cubrir las funcionalidades de forma unitaria y lo más pequeñas y atómicas posibles.

5.2. Sistema de usuarios

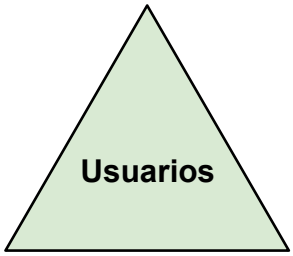

Figura 5.2.1: Sistema de usuarios.

El sistema de usuarios (figura 5.2.1) es la base para controlar el acceso a las dos partes de la aplicación. Existen dos tipos de usuarios, los usuarios normales para acceder a las zonas personales de la interfaz de usuario y los administradores, para acceder a la interfaz de administración. Dado que los usuarios pueden registrarse y acceder mediante las nombradas redes sociales, llamadas internamente proveedores, cada usuario puede tener hasta 3 proveedores asociados (uno por cada proveedor).

5.2.1. Requisitos

Requisitos de administradores:

- Los administradores se crean directamente en la base de datos, para evitar problemas de seguridad.

- Los administradores introducen su dirección de correo electrónico y su contraseña para identificarse.
- Una vez identificados son redireccionados a la sección de administración, donde pueden ver la información los usuarios normales dados de alta en el sistema.

Requisitos de nuevos usuarios:

- Tienen permiso para ver la página de inicio de la aplicación.
- Pueden registrarse a través de un formulario de registro donde se solicita una dirección de correo, un nombre de usuario y una contraseña.
- Si se registran normalmente, reciben un correo con una dirección *URL* con la que confirmar su registro.
- Pueden registrarse a través de las redes sociales *Fatebook*, *Twitter* o *Google+* (proveedores).
- En el caso de registrarse con una red social, se crea directamente un usuario con los datos proporcionados por la red social y se identifica en la aplicación automáticamente.
- En el caso de *Twitter*, no proporciona dirección de correo, por lo que se le pregunta al usuario antes de crearlo.

Requisitos de usuarios ya registrados:

- Pueden identificarse en la aplicación introduciendo su dirección de correo electrónico y su contraseña o con un proveedor ya asociado.
- Al identificarse entran en sesión, lo que les dará permisos cuando existan zonas personales.
- Si no recuerdan su contraseña, pueden solicitar un correo con una dirección *URL* donde establecer una nueva.
- Un usuario bloqueado puede solicitar un correo con su dirección *URL* de confirmación en el caso de haberla perdido.
- Tras cinco intentos erróneos de identificación, el acceso para el usuario se bloquea.
- Un usuario bloqueado puede solicitar un correo con la dirección *URL* donde desbloquear el usuario en el caso de estar bloqueado.
- Un usuario en sesión puede modificar su nombre, su contraseña y añadir o eliminar proveedores.

En la figura 5.2.2 se representa el flujo del usuario a través del sistema de registro e identificación de la aplicación.

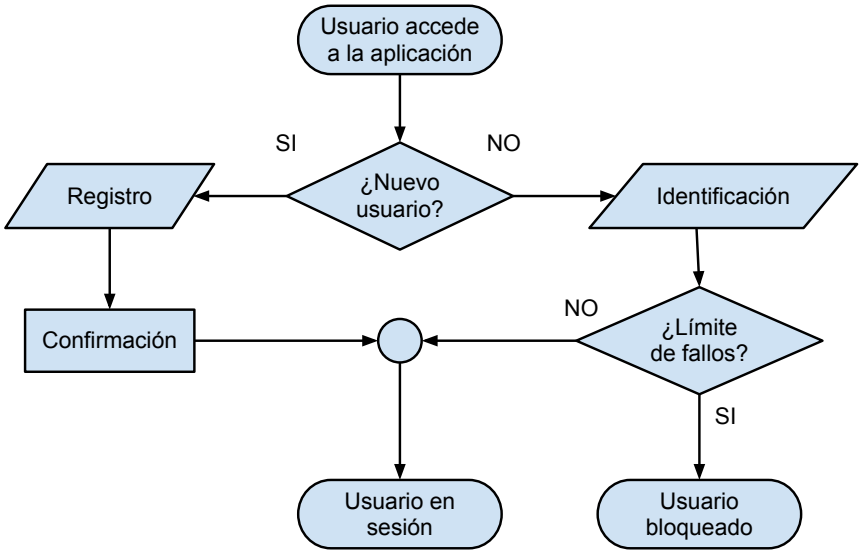

Figura 5.2.2: Flujo de registro e identificación.

5.2.2. Modelos

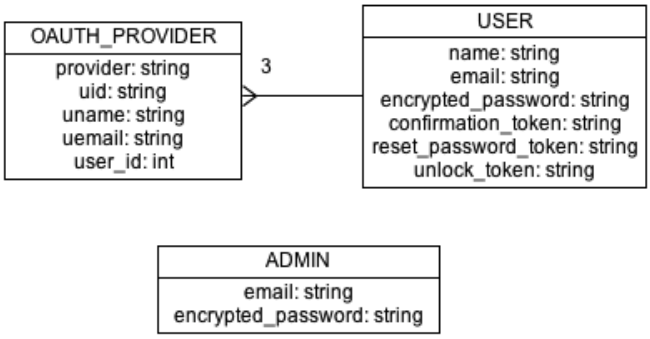

Figura 5.2.3: Modelos del sistema de usuarios.

Modelo de administrador (ADMIN en la figura 5.2.3): información para la identificación de administradores.

Campo	**Descripción**
email	Dirección de correo electrónico válida que identifica al administrador.
encripted_pasword	Contraseña de acceso para el administrador, encriptada para que no sea legible en base de datos.

Cuadro 5.2.1: Modelo de administrador.

Modelo de usuario (USER en la figura 5.2.3): información personal de los usuarios.

Campo	**Descripción**
name	Nombre del usuario para representarlo en la aplicación.
dirección de correo	Dirección de correo electrónico válida que identifica al usuario.
encripted_pasword	Contraseña de acceso para el usuario, encriptada para que no sea legible en base de datos.
confirmation_token	Código alfanumérico para confirmar la creación del usuario.
reset_pasword_token	Código alfanumérico para cambiar la contraseña del usuario.
unlock_token	Código alfanumérico para desbloquear al usuario.

Cuadro 5.2.2: Modelo de usuario.

Modelo de proveedor (OAUTH_PROVIDER en la figura 5.2.3): información de los proveedores asociados a los usuarios para registrarse e identificarse. A través de sus identificadores, se podrá consultar información en los mismos.

Campo	**Descripción**
provider	Nombre del proveedor (*facebook*, *twitter* o *google*).
uid	Identificador del usuario en el proveedor.
uname	Nombre del usuario en el proveedor.
udirección de correo	Dirección de correo del usuario en el proveedor.
user_id	Identificador interno del usuario asociado al proveedor.

Cuadro 5.2.3: Modelo de proveedor.

5.2.3. Vistas

Menú
Menú general para toda la aplicación con los enlaces de las secciones para la navegación. Por ahora son la página principal y la identificación y configuración de usuario.

Identificación de administrador
Solicitud de dirección de correo y contraseña para administradores. Redirige a la página principal de administración o muestra mensaje de error.

Principal de administración
Sección de administrador. Muestra el resumen de datos en la aplicación y la navegación de administración entre los mismos. Por ahora son los usuarios registrados en el sistema y sus datos.

Principal
Página de entrada por defecto a la aplicación. Muestra un texto de bienvenida para el usuario.

Identificación de usuario
Solicitud de dirección de correo y contraseña para administradores. Ofrece la posibilidad de usar proveedores externos.

Registro de usuario
Solicitud de dirección de correo, nombre y contraseña para crear un usuario.

Recuperación de contraseña
Solicitud de dirección de correo para enviar instrucciones de recuperación de contraseña.

Reenvío de confirmación
Solicitud de dirección de correo para reenviar instrucciones de confirmación de usuario.

Desbloqueo de usuario
Solicitud de dirección de correo para enviar instrucciones de desbloqueo de usuario.

Configuración de usuario
Formulario para editar el nombre, la contraseña y los proveedores de un usuario ya creado.

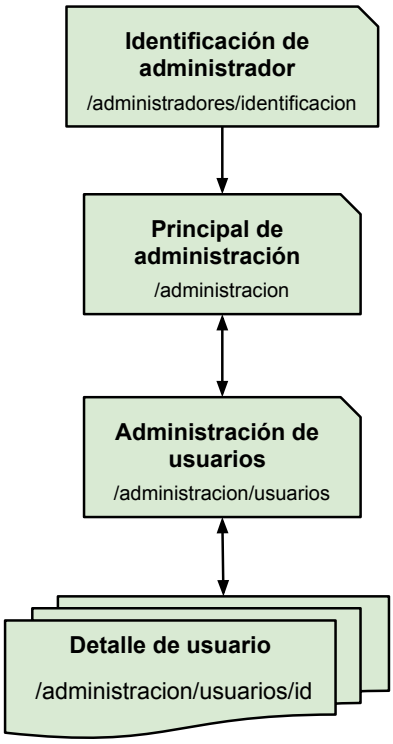

Figura 5.2.4: Navegación de administración en la iteración de usuarios.

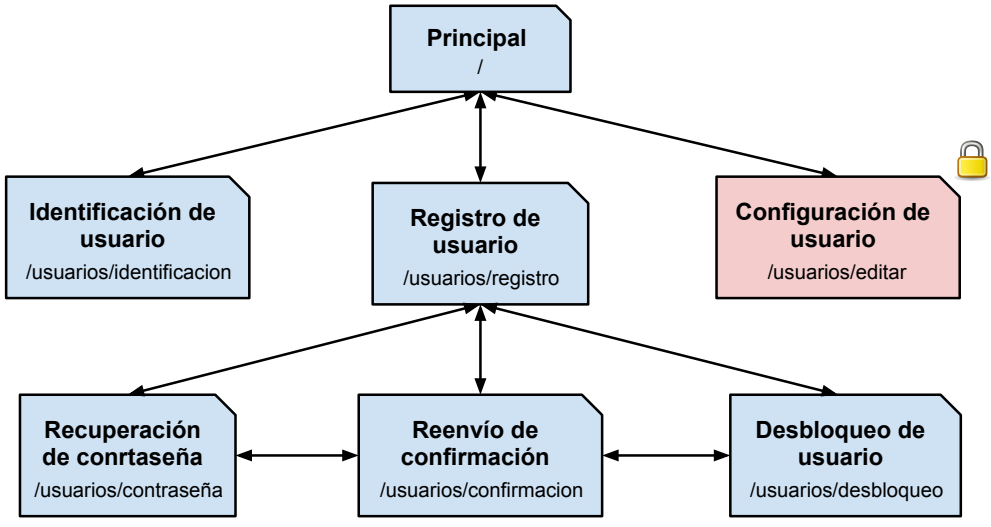

Figura 5.2.5: Navegación de usuario en la iteración de usuarios.

5.2.4. Pruebas

Modelo	Usuario.
Descripción	Creación de un nuevo usuario con datos válidos.
Resultado esperado	El usuario se crea correctamente en la base de datos y se envía un correo electrónico a la dirección de correo.

Cuadro 5.2.4: Prueba de creación de usuario.

Modelo	Usuario.
Descripción	Creación de un usuario con algún campo en blanco.
Resultado esperado	El usuario no se crea y se genera un error.

Cuadro 5.2.5: Prueba de creación de usuario no válido.

Modelo	Usuario.
Descripción	Creación de un usuario con una dirección de correo no válida.
Resultado esperado	El usuario no se crea y se genera un error.

Cuadro 5.2.6: Prueba de creación de usuario con correo inválido.

Modelo	Usuario.
Descripción	Dado un nuevo usuario, activarlo con su código de activación.
Resultado esperado	El usuario se activa y puede identificarse en la página con su contraseña.

Cuadro 5.2.7: Prueba de activación de un usuario.

Modelo	Usuario.
Descripción	Identificarse más de 5 veces con una contraseña inválida.
Resultado esperado	El usuario se bloquea.

Cuadro 5.2.8: Prueba de bloqueo de un usuario.

Modelo	Usuario.
Descripción	Desbloquear un usuario bloqueado con su código de desbloqueo.
Resultado esperado	El usuario se desbloquea y puede identificarse en la aplicación.

Cuadro 5.2.9: Prueba de desbloqueo de un usuario.

Modelo	Usuario.
Descripción	Un usuario se identifica en la web.
Resultado esperado	El usuario entra en sesión.

Cuadro 5.2.10: Prueba de identificación de un usuario.

Modelo	Administrador.
Descripción	Un administrador se identifica en la web.
Resultado esperado	El usuario entra en sesión y pasa a la administración.

Cuadro 5.2.11: Prueba de identificación de un usuario.

Modelo	Usuario.
Descripción	Un usuario accede a la sección de administración.
Resultado esperado	Se deniega el acceso.

Cuadro 5.2.12: Prueba de acceso de un usuario a la administración.

Dado que las pruebas con proveedores de servicios necesitarían la comunicación con proveedores externos y podrían necesitar pasarse en un entorno en el que no estuviesen accesibles o configurados, éstos deben simularse mediante *mocks*[1]. Consultando las documentaciones de los distintos servicios puede prepararse la respuesta esperada ante las peticiones a los mismos.

Modelo	Proveedor de servicio.
Descripción	Un nuevo usuario se crea desde un proveedor de servicio.
Resultado esperado	Se crea un usuario con los datos recibidos del proveedor, está activado y tiene asociado el proveedor.

Cuadro 5.2.13: Prueba de creación de nuevo usuario con proveedor.

[1] Objetos que simulan una respuesta esperada.

En el caso de *Twitter*, por sus términos legales, no proporciona direcciones de correo electrónico a terceros, por lo que necesitamos un caso especial si al crear un usuario no tenemos el correo.

Modelo	Proveedor de servicio.
Descripción	Un nuevo usuario se crea desde un proveedor de servicio que no proporciona correo.
Resultado esperado	El usuario no se crea, se solicita el correo y se continua por el método normal.

Cuadro 5.2.14: Prueba de creación de nuevo usuario con proveedor sin correo.

Modelo	Proveedor de servicio.
Descripción	Un usuario existente se identifica con un proveedor que ya tiene asociado.
Resultado esperado	El usuario se identifica y entra en sesión.

Cuadro 5.2.15: Prueba de identificación de usuario con proveedor existente.

Modelo	Proveedor de servicio.
Descripción	Un usuario existente se identifica con un proveedor que no tiene asociado.
Resultado esperado	Se crea el nuevo proveedor, se asocia al usuario y entra en sesión.

Cuadro 5.2.16: Prueba de identificación de usuario con nuevo proveedor.

Modelo	Proveedor de servicio.
Descripción	Un usuario existente se separa de un proveedor que tiene asociado.
Resultado esperado	Se borra el proveedor asociado al usuario.

Cuadro 5.2.17: Prueba de separación de usuario con un proveedor.

5.3. Administración de ligas

Figura 5.3.1: Administración de ligas.

El sistema de ligas (figura 5.3.1) es el núcleo de la aplicación. La información de las ligas, clubs y jugadores es lo que sustenta al resto del juego. Pueden existir varias ligas y los clubs pueden jugar en una o más ligas. La relación entre los jugadores y los clubs viene determinada por las fichas con la información de ese jugador en ese club (un jugador puede cambiar de demarcación o de valor al cambiar de club).

5.3.1. Requisitos

Requisitos de administradores:

- Pueden crear, editar y consultar los datos de las ligas.
- Pueden crear, editar y consultar los datos de los clubs.
- Pueden crear, editar y consultar los datos de los jugadores.
- Pueden asociar clubs a las ligas.
- Desde la edición de clubs, pueden crear y editar fichas de club para asociar a los jugadores con los clubs.
- Cuando una ficha tiene fecha de salida, se considera que el jugador ya no juega en el club y puede crearse una ficha en otro club.
- Cuando se modifica el valor, el dorsal o la demarcación de la ficha de club de un jugador, se crea un versión nueva para mantener un histórico de fichas.

Requisitos de usuarios:

- Pueden seleccionar en qué liga jugar.
- Pueden consultar la lista de clubs de la liga seleccionada.
- Pueden consultar los detalles de un club para ver su descripción y jugadores.

5.3.2. Modelos

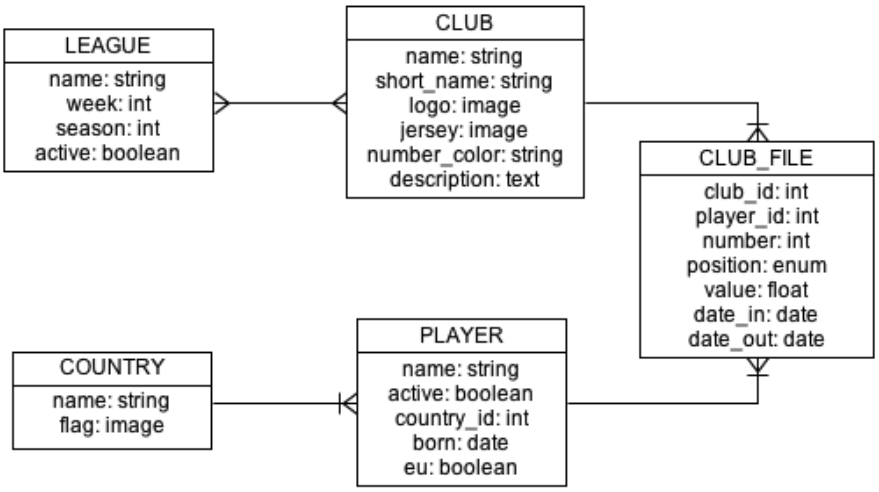

Figura 5.3.2: Modelos de la administración de ligas.

Modelo de liga (LEAGUE en la figura 5.3.2): información de las distintas ligas disponibles para jugar. La temporada y jornada irán determinado los partidos que afectan al juego.

Campo	**Descripción**
name	Nombre de la liga.
week	Jornada que se está disputando en el juego.
season	Temporada que se está disputando en el juego.
active	Determina si la liga está activa o no.

Cuadro 5.3.1: Modelo de liga.

Modelo de club (CLUB en la figura 5.3.2): información de los clubs que participan en las ligas.

Campo	**Descripción**
name	Nombre del club.
short_name	Nombre del club en tres letras.
logo	Imagen del escudo del club.
jersey	Imagen de la camiseta del club.
number_color	Color para el número de los jugadores de la camiseta.
description	Descripción del club.

Cuadro 5.3.2: Modelo de club.

Modelo de jugador (PLAYER en la figura 5.3.2): información de los jugadores que pertenecen a los clubs.

Campo	**Descripción**
name	Nombre del jugador.
active	Determina si el jugador está activa o no.
country_id	Identificador interno del país de nacimiento del jugador.
born	Fecha de nacimiento del jugador.
eu	Determina si el jugador es comunitario o no.

Cuadro 5.3.3: Modelo de jugador.

Modelo de ficha de club (CLUB_FILE en la figura 5.3.2): relaciones entre los jugadores y los clubs. Mientras un jugador está en un club, su ficha no tiene fecha de salida (a partir de ahora la consideraremos abierta). Cuando un jugador abandona un club y entra en otro, se crea una nueva ficha y la antigua se mantiene estableciendo la fecha de la salida (a partir de ahora cerrada), con lo que se almacena la trayectoria del jugador. Además, se mantiene un histórico de los distintos precios que tienen el jugador en una ficha, para poder calcular su valor en cualquier momento del tiempo.

Campo	**Descripción**
club_id	Identificador interno del club al que pertenece el jugador.
player_id	Identificador interno del jugador de la ficha.
number	Dorsal del jugador en el club.
position	Demarcación en el campo de jugador (portero, defensa, centrocampista o delantero).
value	Precio del jugador en el mercado.
date_in	Fecha de entrada del jugador en el club.
date_out	Fecha de salida del jugador del club.

Cuadro 5.3.4: Modelo de ficha de club.

Modelo de país (COUNTRY en la figura 5.3.2): información de los países de los jugadores.

Campo	**Descripción**
name	Nombre del país.
flag	Imagen de la bandera del país.

Cuadro 5.3.5: Modelo de país.

5.3.3. Vistas

Menú
Se añade un selector de ligas al menú, para en el caso de existir varias ligas, poder seleccionar en cuál jugar. Se añade la navegación a la página de clubs.

Administración de ligas
Sección en la administración para las ligas donde poder crearlas, editarlas, consultarlas y asociar los clubs a las distintas ligas.

Administración de clubs
Sección en la administración para los clubs donde poder crearlos, editarlos, consultarlos y crear las fichas de los jugadores.

Administración de jugadores
Sección en la administración para los jugadores donde poder crearlos, editarlos y consultarlos.

Clubs
Sección con el índice de equipos de la liga seleccionada.

Detalle de club
Sección con los detalles de un equipo: su información y sus jugadores.

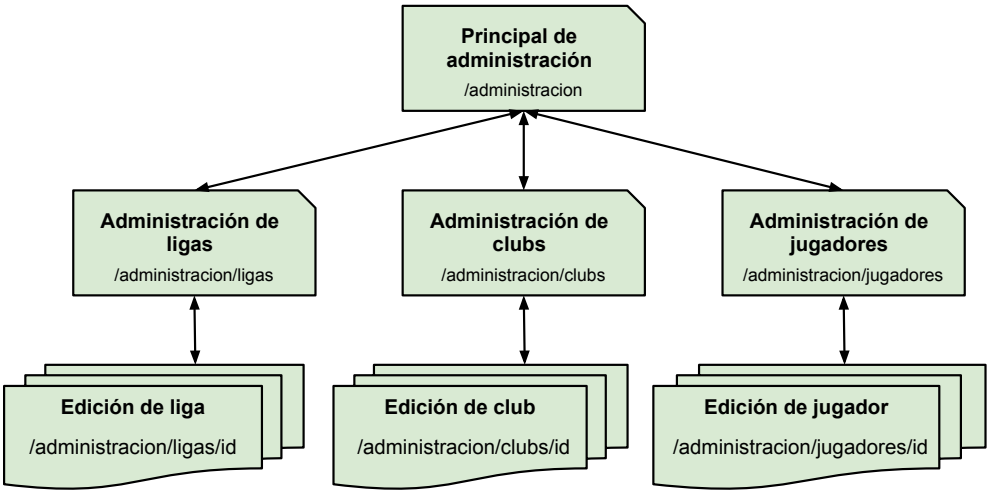

Figura 5.3.3: Navegación de administración en la iteración de ligas.

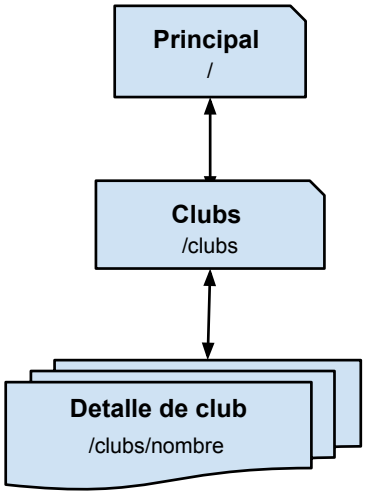

Figura 5.3.4: Navegación de usuario en la iteración de ligas.

5.3.4. Pruebas

Modelo	Liga.
Descripción	Creación de una liga con datos válidos.
Resultado esperado	La liga se crea correctamente en la base de datos.

Cuadro 5.3.6: Prueba de creación de liga.

Modelo	Liga.
Descripción	Creación de una liga con algún campo inválido.
Resultado esperado	La liga no se crea y se genera un error.

Cuadro 5.3.7: Prueba de creación de liga no válida.

Modelo	Club.
Descripción	Creación de un club con datos válidos.
Resultado esperado	El club se crea correctamente en la base de datos.

Cuadro 5.3.8: Prueba de creación de club.

Modelo	Liga.
Descripción	Creación de un club con algún campo inválido.
Resultado esperado	El club no se crea y se genera un error.

Cuadro 5.3.9: Prueba de creación de club no válido.

Modelo	Jugador.
Descripción	Creación de un jugador con datos válidos.
Resultado esperado	El jugador se crea correctamente en la base de datos.

Cuadro 5.3.10: Prueba de creación de jugador.

Modelo	Jugador.
Descripción	Creación de un jugador con algún campo inválido.
Resultado esperado	El jugador no se crea y se genera un error.

Cuadro 5.3.11: Prueba de creación de jugador no válido.

Modelo	País.
Descripción	Creación de un país con datos válidos.
Resultado esperado	El país se crea correctamente en la base de datos.

Cuadro 5.3.12: Prueba de creación de país.

Modelo	País.
Descripción	Creación de un país con algún campo inválido.
Resultado esperado	El país no se crea y se genera un error.

Cuadro 5.3.13: Prueba de creación de país no válido.

Modelo	Ficha de club.
Descripción	Creación de una ficha de club con datos válidos.
Resultado esperado	La ficha de club se crea correctamente en la base de datos.

Cuadro 5.3.14: Prueba de creación de ficha de club.

Modelo	Ficha de club.
Descripción	Creación de una ficha de club con algún campo inválido.
Resultado esperado	La ficha de club no se crea y se genera un error.

Cuadro 5.3.15: Prueba de creación de ficha de club no válida.

Modelo	Ficha de club.
Descripción	Creación de una ficha de club para un jugador que ya tiene una ficha abierta.
Resultado esperado	La ficha de club no se crea y se genera un error. El jugador sólo puede jugar en un club al mismo tiempo y todas sus fichas anteriores deben de estar cerradas.

Cuadro 5.3.16: Prueba de creación de ficha de club con otra abierta.

Modelo	Ficha de club.
Descripción	Creación de una ficha de club para un jugador que ya tiene una ficha cerrada.
Resultado esperado	La ficha de club se crea correctamente en la base de datos.

Cuadro 5.3.17: Prueba de creación de ficha de club con otra cerrada.

Modelo	Ficha de club.
Descripción	Creación de una ficha de club para un jugador que ya tiene una ficha cerrada posteriormente a la nueva.
Resultado esperado	La ficha de club no se crea y se genera un error. El jugador sólo puede jugar en un club al mismo tiempo y con una fecha de entrada anterior a la última de salida o se solaparían en el tiempo.

Cuadro 5.3.18: Prueba de creación de ficha de club con otra cerrada posterior.

Modelo	Ficha de club.
Descripción	Actualización del valor, el dorsal o la demarcación de una ficha de club abierta.
Resultado esperado	La ficha de club se actualiza y se almacena una versión de la ficha con el estado anterior a la actualización.

Cuadro 5.3.19: Prueba de actualización con versionado de ficha de club.

Modelo	Ficha de club.
Descripción	Actualización de cualquier campo que no sea el valor, el dorsal o la demarcación de una ficha de club abierta.
Resultado esperado	La ficha de club se actualiza y no se genera ninguna versión de la ficha.

Cuadro 5.3.20: Prueba de actualización sin versionado de ficha de club.

Modelo	Ficha de club.
Descripción	Actualización del valor, el dorsal o la demarcación de una ficha de club cerrada.
Resultado esperado	La ficha de club no se actualiza y se produce un error. Cuando una ficha de club se cierra no puede modificarse su histórico.

Cuadro 5.3.21: Prueba de actualización de ficha de club cerrada.

Modelo	Club.
Descripción	Consulta de jugadores de un club en una fecha determinada.
Resultado esperado	Se obtiene la lista de jugadores con una fecha de entrada anterior a la fecha indicada y sin fecha de salida o una fecha de salida posterior a la indicada.

Cuadro 5.3.22: Prueba de consulta de jugadores de un club en una fecha.

Modelo	Club.
Descripción	Consulta de jugadores de un club en una demarcación y en una fecha determinada.
Resultado esperado	Se obtiene la lista de jugadores que juegan en una demarcación con una fecha de entrada anterior a la fecha indicada y sin fecha de salida o una fecha de salida posterior a la indicada.

Cuadro 5.3.23: Prueba de consulta de jugadores de un club en una posición y en una fecha.

Modelo	Jugador.
Descripción	Consulta el club de un jugador en una fecha determinada.
Resultado esperado	Se obtiene el club donde el jugador tiene una ficha con una fecha de entrada anterior a la fecha indicada y sin fecha de salida o una fecha de salida posterior a la indicada.

Cuadro 5.3.24: Prueba de consulta de club de un jugador en una fecha.

5.4. Administración de partidos

Figura 5.4.1: Administración de partidos.

Los partidos (figura 5.4.1) son la estructura que organiza el juego jornada a jornada. Pertenecen a una sola liga y tienen siempre dos clubs implicados. Son los que rigen la disponibilidad de compra de los jugadores en función de sus horarios.

5.4.1. Requisitos

Requisitos de administradores:

- Pueden crear, editar y consultar los datos de los partidos.
- Desde la edición de partidos se pueden asociar los clubs que juegan el partido.
- Los partidos se consideran activos por defecto y sólo pueden ser editados en ese estado.
- Los partidos inactivos son ignorados por el juego.
- Para minimizar los errores los partidos tienen que pasar por un estado de revisión antes de considerarse cerrados.

Requisitos de usuarios:

- Pueden consultar la lista de partidos de la una jornada en cualquier temporada de la liga seleccionada.
- Por defecto han de verse los partidos de la jornada actual.

5.4.2. Modelos

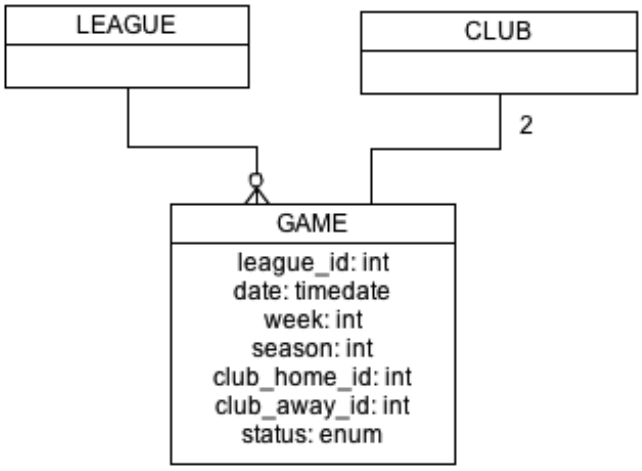

Figura 5.4.2: Modelos de la administración de partidos.

Modelo de partido (GAME en la figura 5.4.2): información de los partidos de una liga.

Campo	**Descripción**
league_id	Identificador interno de la liga a la que pertenece el partido.
club_home_id	Identificador interno del club local del partido.
club_away_id	Identificador interno del club visitante del partido.
date	Fecha y hora a la que comienza el partido.
week	Jornada de la liga en la que se disputa el partido.
season	Temporada de la liga en la que se disputa el partido.
status	Estado del partido (activo, inactivo, evaluado, revisado o cerrado).

Cuadro 5.4.1: Modelo de partido.

Es importante destacar el funcionamiento de los estados de los partidos que se muestra en la figura 5.4.3. Cuando un partido es creado, se considera en estado activo y representa el estado del partido mientras no se ha jugado. En casos especiales en los que el partido, por ejemplo, se aplaza, para que ese partido no interfiera con la jornada se debe pasar a estado inactivo. Cuando un partido se juega y los datos son actualizados, se pasa a evaluado. Antes de alcanzar el estado final de cerrado, el partido pasa por estado revisado. Este paso intermedio ayuda a asegurar una revisión de los datos antes del último estado.

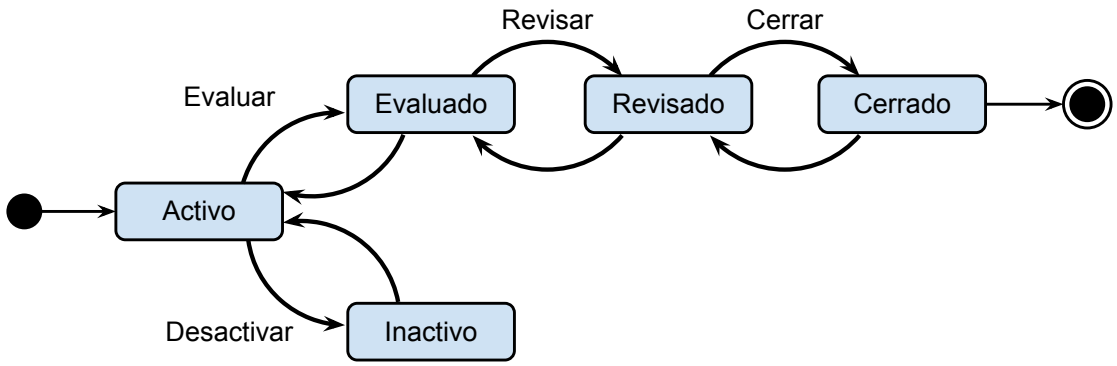

Figura 5.4.3: Estados de partidos.

5.4.3. Vistas

Menú
Se añade la navegación a la sección de partidos de la liga seleccionada.

Administración de partidos
Sección en la administración para los partidos donde poder crearlos, editarlos y consultarlos.

Partidos
Sección para consultar la lista de partidos de una jornada y sus horarios.

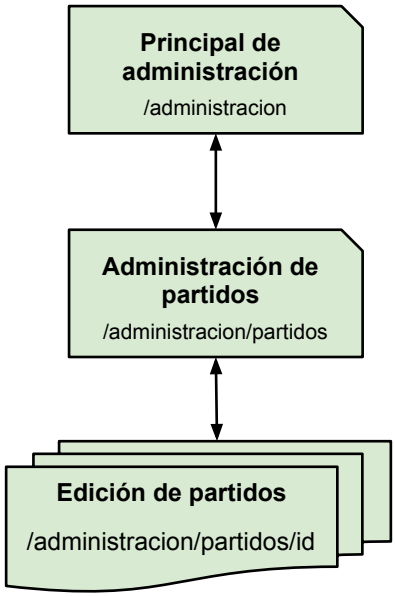

Figura 5.4.4: Navegación de administración en la iteración de partidos.

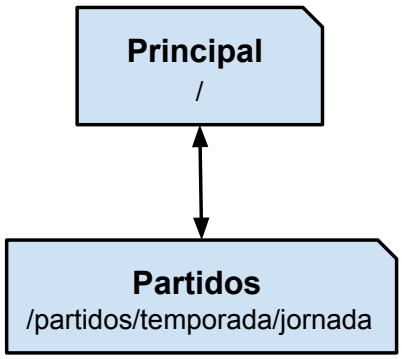

Figura 5.4.5: Navegación de usuario en la iteración de partidos.

5.4.4. Pruebas

Modelo	Partido.
Descripción	Creación de un partido con datos válidos.
Resultado esperado	El partido se crea correctamente en la base de datos.

Cuadro 5.4.2: Prueba de creación de partido.

Modelo	Partido.
Descripción	Creación de un partido con algún campo inválido.
Resultado esperado	El partido no se crea y se genera un error.

Cuadro 5.4.3: Prueba de creación de partido no válida.

Modelo	Partido.
Descripción	Creación de un partido en una liga con clubs de otra liga.
Resultado esperado	El partido no se crea y se genera un error.

Cuadro 5.4.4: Prueba de creación de partido con liga inválida.

Modelo	Partido.
Descripción	Creación de un partido con el mismo club como local y visitante.
Resultado esperado	El partido no se crea y se genera un error.

Cuadro 5.4.5: Prueba de creación de partido con el mismo club.

Modelo	Partido.
Descripción	Consulta de partidos de una liga en una jornada determinada.
Resultado esperado	Se obtiene la lista de partidos de una liga en una jornada determinada.

Cuadro 5.4.6: Prueba de consulta de partidos de una liga en una jornada.

Modelo	Partido.
Descripción	Consulta de partidos de una liga en una temporada determinada.
Resultado esperado	Se obtiene la lista de partidos de una liga en una temporada determinada.

Cuadro 5.4.7: Prueba de consulta de partidos de una liga en una temporada.

Modelo	Partido.
Descripción	Consulta de partidos de una liga en un estado determinado.
Resultado esperado	Se obtiene la lista de partidos de un estado determinado.

Cuadro 5.4.8: Prueba de consulta de partidos de una liga en un estado.

Modelo	Partido.
Descripción	Dado un partido, determinar si un jugador juega en alguno de los clubs del partido o no.
Resultado esperado	Se obtiene verdadero si juega y falso si no.

Cuadro 5.4.9: Prueba de validación de jugador en un partido.

Modelo	Liga.
Descripción	Dado una liga, determinar si la jornada actual no ha empezado todavía.
Resultado esperado	Se obtiene verdadero si todos los partidos de la jornada están activos y falso si no.

Cuadro 5.4.10: Prueba de validación de jornada no empezada.

Modelo	Liga.
Descripción	Dado una liga, determinar si la jornada actual ya ha empezado a disputarse.
Resultado esperado	Se obtiene verdadero si algún partido de la jornada está al menos evaluado y falso si no.

Cuadro 5.4.11: Prueba de validación de jornada empezada.

Modelo	Liga.
Descripción	Dado una liga, determinar si la jornada ha finalizado.
Resultado esperado	Se obtiene verdadero si todos los partidos de la jornada están al menos evaluados y falso si no.

Cuadro 5.4.12: Prueba de validación de jornada no empezada.

5.5. Administración de eventos

Figura 5.5.1: Administración de eventos.

Los eventos (figura 5.5.1) son los distintos acontecimientos que ocurren durante los partidos y que son relevantes para las puntuaciones del juego. Éstos son las alineaciones iniciales de los jugadores o titularidades, las sustituciones, las tarjetas y los goles. Cada elemento tiene sus propias propiedades, pero está claro que gran parte de su comportamiento es común. Además es importante tener un fuerte de sistema validaciones para evitar introducir datos erróneos que puedan desvirtuar el juego.

5.5.1. Requisitos

Requisitos de administradores:

- Los administradores pueden crear, editar y consultar titulares desde los partidos.
- Tiene que haber 11 titulares por club.
- Los administradores pueden crear, editar y consultar sustituciones desde los partidos.
- Sólo puede haber como máximo 3 sustituciones por club.
- Los dos jugadores que participan en una sustitución deben jugar en el mismo club.

- En una sustitución, el jugador que sale debe estar jugando y el jugador que entra no.
- Los administradores pueden crear, editar y consultar tarjetas desde los partidos.
- Sólo puede haber un tipo de cada tarjeta (amarilla, roja y roja directa) por jugador y por partido.
- Un jugador con tarjeta roja, necesita tener una tarjeta amarilla anterior.
- El jugador que recibe la tarjeta debe jugar en ese momento.
- Un jugador está jugando en un determinado momento cuando tiene una titularidad o una sustitución de entrada y no tiene una sustitución de salida o una tarjeta roja o roja directa.
- Los administradores pueden crear, editar y consultar goles desde los partidos.
- Sólo los goles de tipo normal pueden tener un asistente.
- Si un gol tiene asistente, éste debe jugar en el mismo club que el autor.
- El portero que recibe el gol debe calcularse a partir del minuto y el club del autor del gol.
- Los goles en propia puerta deben contabilizarse en el club contrario al del autor.
- Los penaltis fallados o parados no deben contabilizarse en el computo final de goles del partido.
- Los eventos deben crearse en orden cronológico para mantener la consistencia entre ellos.
- Los jugadores que intervienen en un evento deben de jugar en alguno de los clubs del partido.

Requisitos de usuarios:

- Pueden consultar los detalles de los partidos desde la lista de partidos.

5.5.2. Modelos

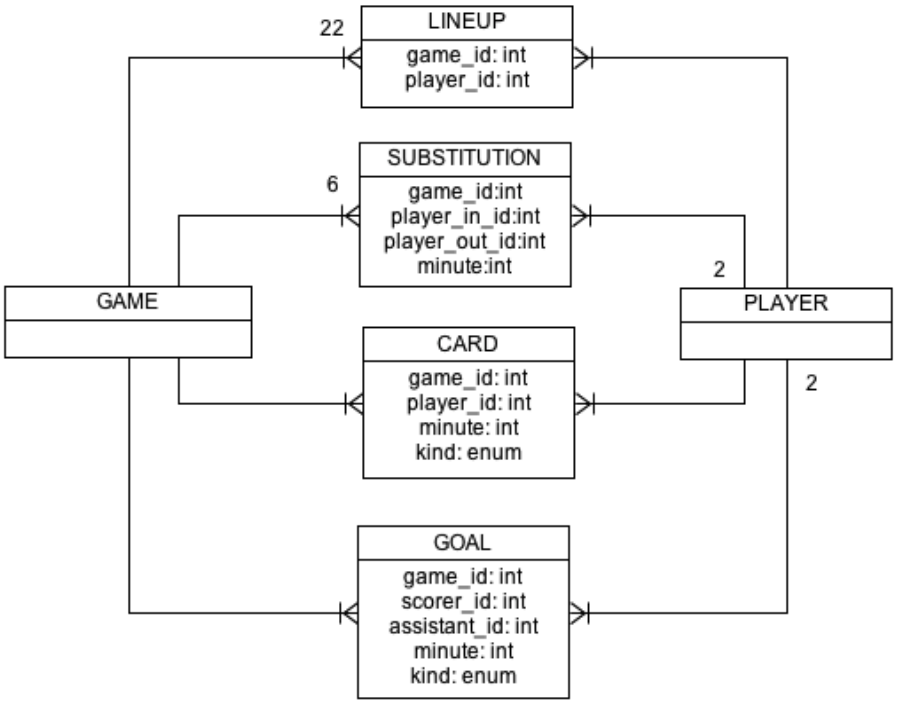

Figura 5.5.2: Modelos de la administración de eventos.

Modelo de titular (LINEUP en la figura 5.5.2): eventos que determinan los jugadores que juegan desde el principio de un partido.

Campo	**Descripción**
game_id	Identificador interno del partido en el que ocurre el evento.
player_id	Identificador interno del jugador titular.

Cuadro 5.5.1: Modelo de titular.

Modelo de sustitución (SUSTITUTION en la figura 5.5.2): evento que se produce cuando un jugador es sustituido por otro durante un partido.

Campo	Descripción
game_id	Identificador interno del partido en el que ocurre evento.
player_in_id	Identificador interno del jugador que entra al partido.
player_out_id	Identificador interno del jugador que sale del partido.
minute	Minuto en el que sucede la sustitución.

Cuadro 5.5.2: Modelo de sustitución.

Modelo de trajeta (CARD en la figura 5.5.2): evento que se produce cuando un jugador es amonestado con una tarjeta durante un partido.

Campo	Descripción
game_id	Identificador interno del partido en el que ocurre el evento.
player_id	Identificador interno del jugador al que le sacan la tarjeta.
minute	Minuto en el que se saca la tarjeta.
kind	Tipo de tarjeta que se saca (amarilla, roja o roja directa).

Cuadro 5.5.3: Modelo de tarjeta.

Modelo de gol (GOAL en la figura 5.5.2): evento que se produce cuando un jugador marca un gol o falla un penalti durante un partido. Los goles en propia puerta se contabilizan para el club contrario al del autor. Los penaltis fallados no se contabilizan como goles en el partido. El portero se calcula automáticamente en función del autor, el tipo de gol y el minuto.

Campo	Descripción
game_id	Identificador interno del partido en el que ocurre el evento.
scorer_id	Identificador interno del jugador autor del gol.
assistant_id	Identificador interno del jugador que hace el pase de gol.
goalkeeper_id	Identificador interno del portero que encaja el gol.
minute	Minuto en el que se marca el gol.
kind	Tipo de tarjeta que se saca (normal, en propia, penalti, penalti fallado o penalti parado).

Cuadro 5.5.4: Modelo de gol.

Extensión de modelos de evento de partido (GAME_EVENT): dado que todos los modelos comparten una serie de funcionalidades comunes, para aplicar el principio *DRY* (citado en el apartado sobre *Ruby on Rails*), se crea una extensión para todos estos modelos para realizar las definiciones una sola vez:

- Todos los eventos tienen un equipo asociado.
- Todos los eventos tienen al menos un jugador asociado.
- Todos los eventos ocurren en un minuto determinado (las titularidades no tienen un atributo minuto, pero ocurren en el minuto 0).

Todas las validaciones y métodos que surjan en las pruebas pueden ser especificadas una única vez dentro de esta extensión, y todos los modelos harán uso de ellas.

5.5.3. Vistas

Edición de partido
Dentro de la vista de edición de partidos ya creada en la anterior iteración, se añade la posibilidad de crear y editar los nuevos modelos asociados a los partidos: titularidades, sustituciones, tarjetas y goles.

Detalle de partido
Nueva vista para consultar los detalles de los partidos. Se muestra el resultado del partido y la lista de eventos ordenados cronológicamente.

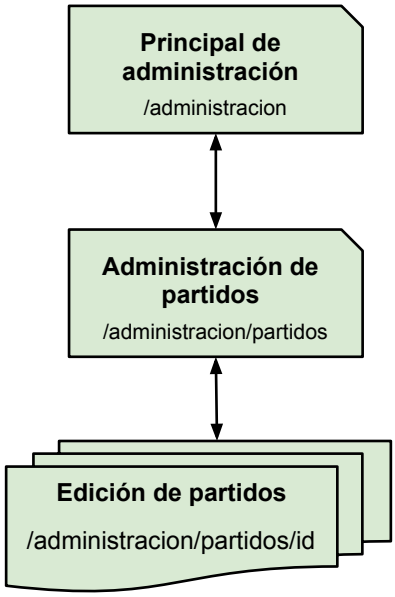

Figura 5.5.3: Navegación de administración en la iteración de eventos.

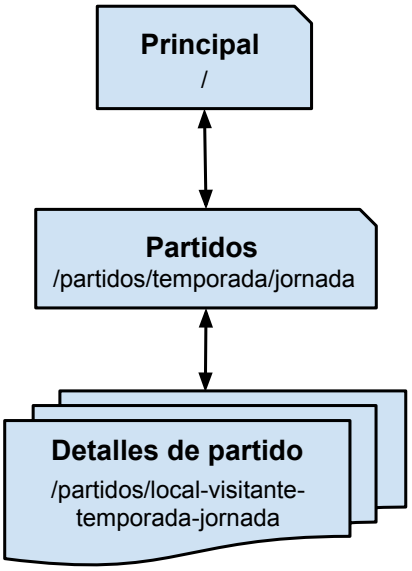

Figura 5.5.4: Navegación de usuario en la iteración de eventos.

5.5.4. Pruebas

Como se ha dicho en el apartado de modelos, las validaciones comunes se probarán sobre la extensión *Evento de partido*.

Modelo	Titularidad.
Descripción	Creación de una titularidad con datos válidos.
Resultado esperado	La titularidad se crea correctamente en la base de datos.

Cuadro 5.5.5: Prueba de creación de titularidad.

Modelo	Titularidad.
Descripción	Creación de una titularidad con algún campo inválido.
Resultado esperado	La titularidad no se crea y se genera un error.

Cuadro 5.5.6: Prueba de creación de titularidad no válida.

Modelo	Titularidad.
Descripción	Creación de una titularidad en un club que ya tiene 11 titulares en el partido.
Resultado esperado	La titularidad no se crea y se genera un error.

Cuadro 5.5.7: Prueba de creación de titularidad sobre el límite.

Modelo	Sustitución.
Descripción	Creación de una sustitución con datos válidos.
Resultado esperado	La sustitución se crea correctamente en la base de datos.

Cuadro 5.5.8: Prueba de creación de sustitución.

Modelo	Sustitución.
Descripción	Creación de una sustitución con algún campo inválido.
Resultado esperado	La sustitución no se crea y se genera un error.

Cuadro 5.5.9: Prueba de creación de sustitución no válida.

Modelo	Sustitución.
Descripción	Creación de una sustitución en un club que ya tiene 3 sustituciones en el partido.
Resultado esperado	La sustitución no se crea y se genera un error.

Cuadro 5.5.10: Prueba de creación de sustitución sobre el límite.

Modelo	Tarjeta.
Descripción	Creación de una tarjeta con datos válidos.
Resultado esperado	La tarjeta se crea correctamente en la base de datos.

Cuadro 5.5.11: Prueba de creación de tarjeta.

Modelo	Tarjeta.
Descripción	Creación de una tarjeta con algún campo inválido.
Resultado esperado	La tarjeta no se crea y se genera un error.

Cuadro 5.5.12: Prueba de creación de tarjeta no válida.

Modelo	Tarjeta.
Descripción	Creación de una segunda tarjeta para un jugador que ya tiene otra del mismo tipo.
Resultado esperado	La tarjeta no se crea y se genera un error.

Cuadro 5.5.13: Prueba de creación de tarjeta sobre el límite.

Modelo	Tarjeta.
Descripción	Creación de una tarjeta roja para un jugador que no tiene una amarilla anterior.
Resultado esperado	La tarjeta no se crea y se genera un error.

Cuadro 5.5.14: Prueba de creación de roja tarjeta sin amarilla.

Modelo	Gol.
Descripción	Creación de un gol con datos válidos.
Resultado esperado	El gol se crea correctamente en la base de datos.

Cuadro 5.5.15: Prueba de creación de gol.

Modelo	Gol.
Descripción	Creación de un gol con algún campo inválido.
Resultado esperado	El gol no se crea y se genera un error.

Cuadro 5.5.16: Prueba de creación de gol no válido.

Modelo	Gol.
Descripción	Creación de un gol distinto al tipo normal con asistente.
Resultado esperado	El gol no se crea y se genera un error.

Cuadro 5.5.17: Prueba de creación de gol con asistente.

Modelo	Evento de partido.
Descripción	Creación de un evento para un jugador que no está jugando en ningún club del partido.
Resultado esperado	El evento no se crea y se genera un error.

Cuadro 5.5.18: Prueba de creación de evento de jugador inválido.

Modelo	Evento de partido.
Descripción	Creación de un evento con dos jugadores que no juegan en el mismo club.
Resultado esperado	El evento no se crea y se genera un error.

Cuadro 5.5.19: Prueba de creación de evento con jugadores en distinto club.

Modelo	Partido.
Descripción	Dado un jugador, un partido y un minuto, saber si el jugador estaba jugando el partido.
Resultado esperado	Determina en función de los eventos del partido si el jugador está jugando en el minuto indicado. Un jugador está jugando cuando tiene una titularidad o una sustitución de entrada y no tiene una sustitución de salida o una tarjeta roja o roja directa.

Cuadro 5.5.20: Prueba de consulta de jugador jugando un partido.

Modelo	Partido.
Descripción	Consulta de los eventos de un partido según el tipo.
Resultado esperado	Lista de los eventos del partido ordenada cronológicamente.

Cuadro 5.5.21: Prueba de consulta de eventos de un partido.

Modelo	Partido.
Descripción	Calcular el resultado de un partido.
Resultado esperado	En función de los tipos de goles que tenga el partido, calcular el resultado final del mismo. Goles en propia en el club contrario al jugador. Penaltis fuera o fallados no contados. Y el resto de goles para el club del autor.

Cuadro 5.5.22: Prueba de consulta de resultado de un partido.

5.6. Creación de equipos

Figura 5.6.1: Creación de equipos.

Los equipos (figura 5.6.1) son los contenidos de la aplicación generados por los usuarios. Utilizando los datos disponibles sobre los clubs y los jugadores de una liga, han de formar sus equipos siguiendo una serie de restricciones. No requieren ningúna sección de administración ya que son únicamente gestionados por sus dueños.

5.6.1. Requisitos

Requisitos de usuarios:

- Sólo un usuario identificado puede crear equipos.
- Pueden crear al menos dos equipos con un nombre distintivo por liga y temporada.
- Un equipo recién creado está inactivo.
- Una vez creado el equipo el usuario puede comprar jugadores.
- El usuario puede buscar jugadores entre todos los activos de la liga actual.
- Un jugador no puede ser comprado si no se dispone del presupuesto necesario.
- Un jugador no puede ser comprado a partir de una hora antes del partido que juega en la jornada actual.
- Una vez el equipo está activo el usuario puede realizar 3 cambios de jugadores cumpliendo siempre las restricciones del equipo.

Un usuario puede activar un equipo cuando éste cumple que:

- Tiene 11 jugadores.
- Tiene un portero.
- Tiene al menos 3 defensas y a lo sumo 5.
- Tiene al menos 3 centrocampistas y a lo sumo 4.
- Tiene al menos 1 delantero y a lo sumo 3.
- No tiene más de 3 jugadores extracomunitarios.
- No tiene más de 3 jugadores del mismo club.

5.6.2. Modelos

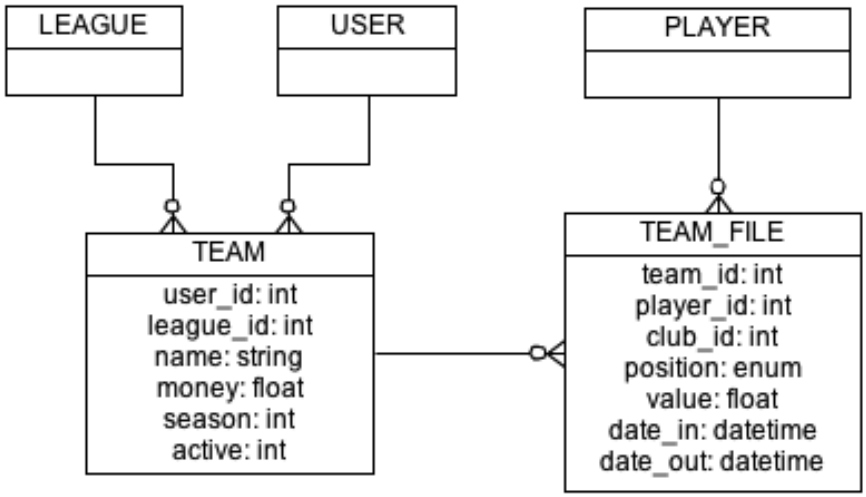

Figura 5.6.2: Modelos de la creación de equipos.

Modelo de equipo (TEAM en la figura 5.6.2): información de los equipos de los usuarios.

Campo	Descripción
user_id	Identificador interno del usuario propietario del equipo.
league_id	Identificador interno de la liga en la que participa el equipo.
name	Nombre del equipo.
money	Dinero disponible para comprar jugadores.
season	Temporada en la que participa el equipo.
active	Determina si un equipo está activo o no.

Cuadro 5.6.1: Modelo de equipo.

Modelo de ficha de equipo (TEAM_FILE en la figura 5.6.2): relaciones entre los jugadores y los equipos de los usuarios. Mientras un jugador está en un equipo, su ficha no tiene fecha de salida. Cuando un jugador abandona un equipo, la ficha se mantiene, con lo que se almacena la trayectoria del jugador. La ficha de equipo copia los datos de posición y valor de la ficha de club a la que está asociada el jugador en el momento de la compra. De esta forma, las restricciones del equipo siempre se mantendrán con los estados iniciales. Si el jugador real cambia de posición, valor o club, esto no afecta a las restricciones del equipo del usuario, evitando así estados incongruentes del equipo.

Campo	Descripción
team_id	Identificador interno del equipo al que pertenece el jugador.
player_id	Identificador interno del jugador de la ficha.
club_id	Identificador interno del club del jugador en el momento de la compra.
position	Posición del jugador en el momento de la compra.
value	Precio del jugador en el momento de la compra.
date_in	Fecha de entrada del jugador en el club.
date_out	Fecha de salida del jugador del club.

Cuadro 5.6.2: Modelo de ficha de equipo.

Extensión de modelos ficha de jugador (PLAYER_FILE): dado que las fichas de clubs y las nuevas fichas de equipos comparten gran parte de las funcionalidades, al igual que con los eventos, vamos a extender estos comportamientos en fichas de jugadores:

- Las dos fichas relacionan un jugador y un club.
- Las dos fichas tienen posición, valor, fecha de entrada y fecha de salida.

Todas las validaciones y métodos que surjan en las pruebas, pueden ser especificadas una única vez dentro de esta extensión y los dos modelos harán uso de ellas.

5.6.3. Vistas

En esta iteración, no es necesario cambios ni nuevas vistas de administración ya que se trata exclusivamente de contenido de los usuarios.

Menú
Se añade una sección para la navegación de equipos, sólo disponible para los usuarios identificados.

Equipos del usuario
La página muestra la lista de equipos del usuario identificado y un resumen de los datos de cada equipo.

Nuevo equipo
Página con un formulario para solicitar al usuario el nombre del equipo a crear.

Edición de equipo
Es la vista más importante del usuario donde realiza todas las opciones de su equipo. Para no hacer el proceso de compra de jugadores complicado mediante el uso de peticiones asíncronas al servidor, se integra la búsqueda, compra y venta de jugadores en esta página.

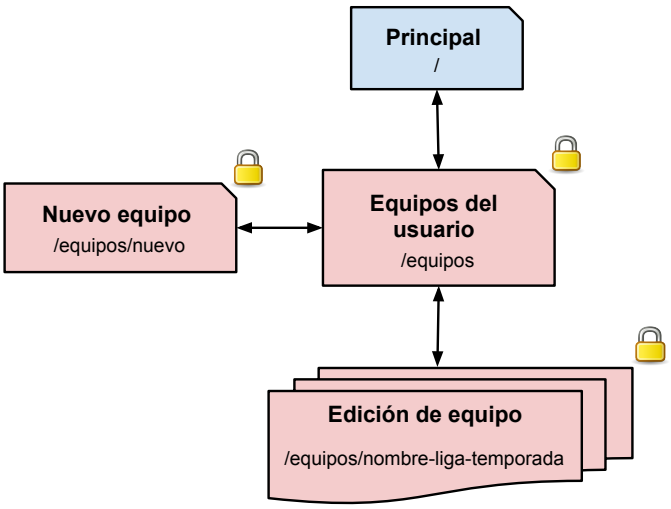

Figura 5.6.3: Navegación de usuario en la iteración de equipos.

5.6.4. Pruebas

Modelo	Equipo.
Descripción	Creación de un equipo con datos válidos.
Resultado esperado	El equipo se crea correctamente en la base de datos.

Cuadro 5.6.3: Prueba de creación de equipo.

Modelo	Equipo.
Descripción	Creación de un equipo con algún campo inválido.
Resultado esperado	El equipo no se crea y se genera un error.

Cuadro 5.6.4: Prueba de creación de equipo no válida.

Modelo	Equipo.
Descripción	Creación de un equipo por un jugador que ya tiene 2 equipos en la misma liga la misma temporada.
Resultado esperado	El equipo no se crea y se genera un error.

Cuadro 5.6.5: Prueba de creación de equipo sobre el límite.

Modelo	Equipo.
Descripción	Activación de un equipo que no tiene todas las posiciones ocupadas.
Resultado esperado	El equipo no se activa y se genera un error.

Cuadro 5.6.6: Prueba de activación de equipo incompleto.

Modelo	Ficha de equipo.
Descripción	Creación de una ficha de equipo con datos válidos.
Resultado esperado	La ficha de equipo se crea correctamente en la base de datos.

Cuadro 5.6.7: Prueba de creación de ficha de equipo.

Modelo	Ficha de equipo.
Descripción	Creación de una ficha de equipo con algún campo inválido.
Resultado esperado	La ficha de equipo no se crea y se genera un error.

Cuadro 5.6.8: Prueba de creación de ficha de equipo no válida.

Modelo	Ficha de equipo.
Descripción	Creación de una ficha de equipo en un equipo que ya tiene 11 jugadores.
Resultado esperado	La ficha de equipo no se crea y se genera un error.

Cuadro 5.6.9: Prueba de creación de una ficha de equipo sobre el límite.

Modelo	Ficha de equipo.
Descripción	Creación de una ficha de equipo de un valor mayor al presupuesto restante del equipo.
Resultado esperado	La ficha de equipo no se crea y se genera un error.

Cuadro 5.6.10: Prueba de creación de una ficha de equipo sobre el límite de presupuesto.

Modelo	Ficha de equipo.
Descripción	Creación de una ficha de equipo en un equipo que ya tiene todos los jugadores en su posición.
Resultado esperado	La ficha de equipo no se crea y se genera un error.

Cuadro 5.6.11: Prueba de creación de una ficha de equipo sobre el límite de posición.

Modelo	Ficha de equipo.
Descripción	Creación de una ficha de equipo de un jugador en un club en un equipo que ya tiene 3 jugadores del mismo club.
Resultado esperado	La ficha de equipo no se crea y se genera un error.

Cuadro 5.6.12: Prueba de creación de una ficha de equipo sobre el límite de clubs.

Modelo	Ficha de equipo.
Descripción	Creación de una ficha de equipo de un jugador extracomunitario en un equipo que ya tiene 3 jugadores extracomunitarios.
Resultado esperado	La ficha de equipo no se crea y se genera un error.

Cuadro 5.6.13: Prueba de creación de una ficha de equipo sobre el límite.

Modelo	Ficha de equipo.
Descripción	Creación de una ficha de equipo de un jugador cuyo partido de la jornada comienza al menos dentro de una hora.
Resultado esperado	La ficha de equipo no se crea y se genera un error.

Cuadro 5.6.14: Prueba de creación de una ficha de equipo sobre el límite de tiempo.

5.7. Puntuaciones

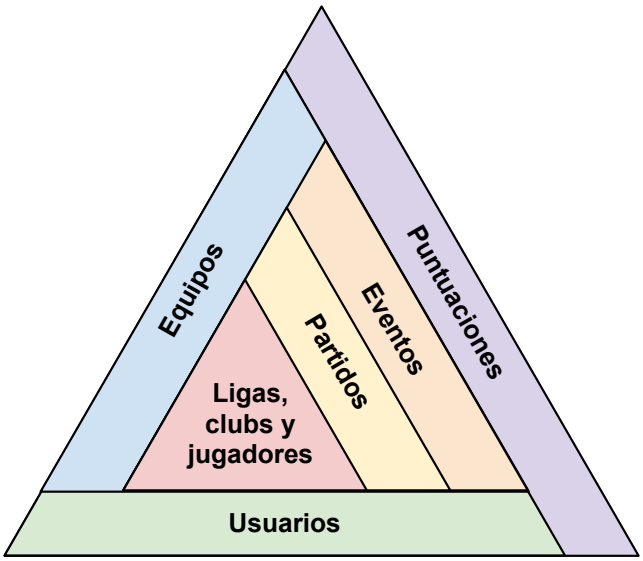

Figura 5.7.1: Puntuaciones.

El sistema de puntuación (figura 5.7.1) da sentido al juego. Dicho sistema aplica las reglas sobre toda la información recogida por la aplicación para calcular automáticamente las puntuaciones de los jugadores en función de sus actuaciones y clasifica los equipos de los jugadores en función de la suma de puntos acumulada por sus jugadores.

Posee, además, la flexibilidad de cambiar los datos de los eventos en cualquier momento (por si hay errores) y los procesos automáticos restauran los puntos adecuadamente. Los equipos de los usuarios consultan siempre sus jugadores para el cálculo de puntos, por lo que la información está actualizada en todo momento.

5.7.1. Requisitos

Requisitos de administradores:

- Al crear los distintos eventos de los partidos, se aplican automáticamente las reglas de puntuación y se acumulan en cada jugador correspondiente los puntos necesarios.

- Al calcular los puntos de los jugadores se acumulan también estadísticas sobre los eventos asociados a ellos (goles marcados, goles encajados y asistencias entre otros).
- Al modificar los datos de un evento, los puntos de los jugadores asociados se actualizan.
- Al modificar los jugadores de un evento, los puntos vuelven a asociar a los jugadores adecuados.
- Al eliminar un evento, se eliminan los puntos de los jugadores asociados.
- Cuando un partido es evaluado, se calculan los puntos asociados a reglas que necesitan todos los eventos del partido introducido (como determinar qué club gana el partido).
- Tras cerrar todos los partidos, la liga puede pasar a la siguiente jornada.

Requisitos de usuarios:

- Pueden consultar las puntuaciones de los equipos en una clasificación ordenada por puntos.
- Las clasificaciones pueden verse por temporada o por jornada.
- Dentro de cada clasificación pueden verse las puntuaciones sólo con los amigos de las redes sociales con las que el usuario está asociado.

5.7.2. Modelos

Para poder calcular las puntuaciones de los equipos en las distintas categorías, se almacena un sumatorio de los puntos generados por un jugador en cada partido. Los puntos de los equipos se calculan en función de los puntos de sus jugadores en cada jornada.

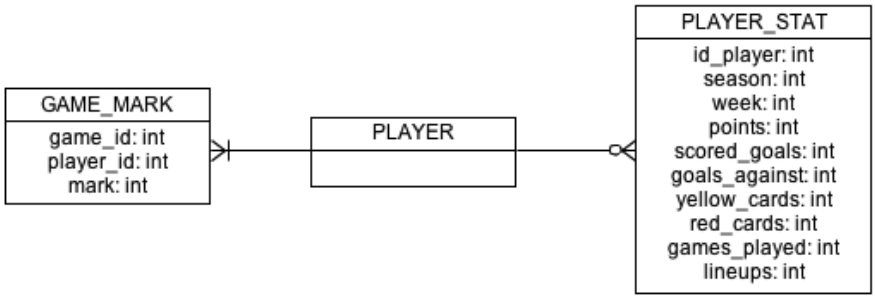

Figura 5.7.2: Modelos de las puntuaciones.

Modelo de puntuación (GAME_MARK en la figura 5.7.2): puntuación objetiva en función de la participación de un jugador en un partido.

Campo	Descripción
game_id	Identificador interno del partido en el que ocurre evento.
player_id	Identificador interno del jugador titular.
mark	Puntos obtenidos por el jugador en el partido (1, 2 o 3).

Cuadro 5.7.1: Modelo de puntuación.

Modelo de estadística (PLAYER_STAT en la figura 5.7.2): inforación resumen de las acciones de un jugador en un partido.

Campo	Descripción
game_id	Identificador interno del partido en el ocurren las estadísticas.
player_id	Identificador interno del jugador que posee las estadísticas.
points	Puntos acumulados por el jugador en ese partido.
goals_scored	Goles marcados por el jugador en el partido.
assists	Asistencias de gol realizadas por el jugador en el partido.
goals_conceded	Goles encajados por el jugador en el partido.
yellow_cards	Tarjetas amarillas sacadas al jugador en el partido.
red_cards	Tarjetas rojas (directas e indirectas) sacadas al jugador en el partido.
lineups	Determina si el jugador ha jugado de titular o no..
games_played	Determina si el jugador ha jugado el partido o no.
minutes_played	Cantidad de minutos del partido jugados por el jugador.

Cuadro 5.7.2: Modelo de estadística.

5.7.3. Vistas

En esta iteración no son necesarios cambios ni nuevas vistas de administración, ya que todos los procesos se realizan automáticamente con la interfaz ya existente.

Menú
Se añade la navegación a la sección de puntuaciones de la liga seleccionada.

Principal
La página principal se actualiza con la clasificación de los mejores 5 jugadores y los mejores 5 equipos de la jornada.

Detalles de jugadores

Ventana modal accesible desde cualquier parte de la página que muestra los detalles de los jugadores, como estadísticas y los puntos que consigue cada jornada.

Edición de equipo
Ahora que se obtienen los puntos de los jugadores, se muestran los puntos de los equipos por jornada.

Puntuaciones por temporada
Página para mostrar las puntuaciones de los equipos de los usuarios ordenadas en forma de clasificación, donde el primero es el que más puntos tiene sumando los puntos de sus jugadores durante todas las jornadas. Si el usuario está asociado con alguna red social, pueden mostrarse sólo los equipos de los amigos en esa red social.

Puntuaciones por jornada
Página para mostrar las puntuaciones de los equipos de los usuarios ordenadas en forma de clasificación, donde el primero es el que más puntos tiene sumando los puntos de sus jugadores durante una jornada indicada. Si el usuario está asociado con alguna red social, pueden mostrarse sólo los equipos de los amigos en esa red social.

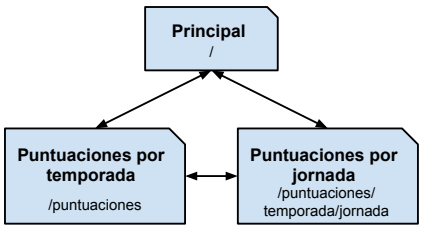

Figura 5.7.3: Navegación de usuario en la iteración de puntuaciones.

5.7.4. Pruebas

Modelo	Evento de partido.
Descripción	Creación de un evento de partido válido.
Resultado esperado	Se crean (o actualizan si ya tienen) las estadísticas de los jugadores asociados al evento para la jornada y se acumulan los puntos y estadísticas adecuados al evento.

Cuadro 5.7.3: Prueba de cálculo de estadísticas al crear un evento.

Modelo	Evento de partido.
Descripción	Modificación de datos de un evento de partido.
Resultado esperado	Se borran las estadísticas relativas al evento de los jugadores asociados y se vuelven a calcular con los nuevos datos del evento.

Cuadro 5.7.4: Prueba de cálculo de estadísticas al modificar el evento.

Modelo	Evento de partido.
Descripción	Modificación de jugador de un evento de partido.
Resultado esperado	Se borran las estadísticas relativas al evento asociado al jugador modificado, y se crean (o actualizan si ya tienen)las estadísticas del nuevo jugador.

Cuadro 5.7.5: Prueba de cálculo de estadísticas al cambiar el jugador del evento.

Modelo	Evento de partido.
Descripción	Eliminación de un evento asociado a un partido.
Resultado esperado	Se borran las estadísticas relativas al evento a los jugadores asociados al mismo.

Cuadro 5.7.6: Prueba de cálculo de estadísticas al borrar un evento.

Modelo	Jugador.
Descripción	Consulta de clasificación de jugadores de una temporada determinada.
Resultado esperado	Se obtiene la lista de jugadores ordenados de mayor a menor por el cálculo de puntos acumulado durante la temporada indicada.

Cuadro 5.7.7: Prueba de consulta de clasificación de jugadores por temporada.

Modelo	Jugador.
Descripción	Consulta de clasificación de jugadores de una jornada determinada.
Resultado esperado	Se obtiene la lista de jugadores ordenados de mayor a menor por el cálculo de puntos obtenido en la jornada indicada.

Cuadro 5.7.8: Prueba de consulta de clasificación de jugadores por temporada.

Modelo	Jugador.
Descripción	Consulta de puntos de un jugador jornada a jornada en una temporada determinada.
Resultado esperado	Se obtiene la lista de los puntos obtenidos en cada jornada de la temporada indicada.

Cuadro 5.7.9: Prueba de puntos de un jugador jornada a jornada.

Modelo	Equipo.
Descripción	Consulta de clasificación de equipos de una temporada determinada.
Resultado esperado	Se obtiene la lista de equipos ordenados de mayor a menor por el cálculo de puntos acumulado por sus jugadores durante la temporada indicada.

Cuadro 5.7.10: Prueba de consulta de clasificación de equipos por temporada.

Modelo	Equipo.
Descripción	Consulta de clasificación de equipos de una jornada determinada.
Resultado esperado	Se obtiene la lista de equipos ordenados de mayor a menor por el cálculo de puntos obtenidos por sus jugadores en la jornada indicada.

Cuadro 5.7.11: Prueba de consulta de clasificación de equipos por jornada.

Modelo	Equipo.
Descripción	Consulta de puntos de un jugador jornada a jornada en una temporada determinada.
Resultado esperado	Se obtiene la lista de los puntos obtenidos en cada jornada de la temporada indicada.

Cuadro 5.7.12: Prueba de puntos de un equipo jornada a jornada.

Modelo	Liga.
Descripción	Cerrar la jornada actual de la liga para que el juego avance a la siguiente.
Resultado esperado	Todos los partidos con estado revisado se pasan a estado cerrado y se suma uno al valor de la jornada de la liga siempre que no sea la última.

Cuadro 5.7.13: Prueba de cerrar jornada de liga.

Modelo	Liga.
Descripción	Cerrar la jornada actual de la liga con algún partido con estado distinto a revisado.
Resultado esperado	No se hace ningún cambio en los datos y se genera un error.

Cuadro 5.7.14: Prueba de cerrar jornada de liga sin revisar.

Capítulo 6

Producto actual

Tras el desarrollo de estas iteraciones el proyecto se encuentra en un estado completamente funcional e integrado, en el que los usuarios ya pueden participar en el juego con todas las características descritas en un principio. Cabe indicar que éste no es el estado final del juego. El desarrollo ágil es un desarrollo orgánico en el que las funcionalidades van surgiendo, superponiéndose y ampliándose unas a otras. Como veremos en el siguiente apartado, podemos plantear nuevas iteraciones con ideas y necesidades que han surgido durante el desarrollo.

La aplicación se encuentra actualmente publicada en Internet y puede accederse con un navegador web en `www.onleague.org`. Ya existen usuarios reales jugando, pero por el momento el acceso está limitado por autentificación básica del navegador y para acceder hay que introducir las siguientes credenciales:

Usuario: shigeru

Contraseña: miyamoto

Esta autentificación no tiene que ver con la aplicación, únicamente es para obtener acceso al servidor web. Una vez dentro, el funcionamiento de la aplicación es el descrito en esta documentación (creación de usuario, equipos...). En el momento en el que la aplicación se haga completamente pública esta autentificación ya no será necesaria.

El código de la aplicación está disponible públicamente para ser consultado o descargado en *GitHub* desde `https://github.com/rsierra/onLeague`. Además, un usuario podría hacer modificaciones y solicitar que sean añadidas a la aplicación. Desde la página web puede verse todo el historial de cambios en la aplicación y suscribirse a notificaciones de actualización. Además, dispone de una herramienta de gestión de tareas (usada durante el desarrollo) donde los usuarios pueden, por ejemplo, solicitar cambios, aportar ideas o reportar fallos.

Como resumen y para ver más fácilmente el estado actual del proyecto, se han unificado, según el tipo en las figuras 6.0.1, 6.0.2, 6.0.3 y 6.0.4, todos los diagramas utilizados en las distintas iteraciones:

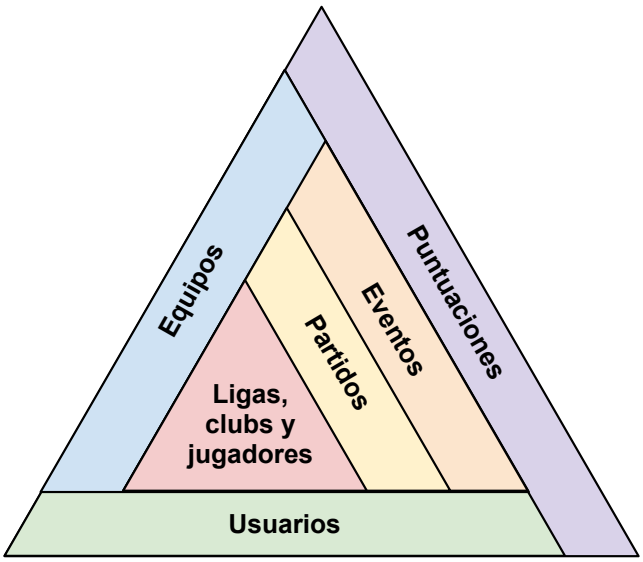

Figura 6.0.1: Estado actual de iteraciones.

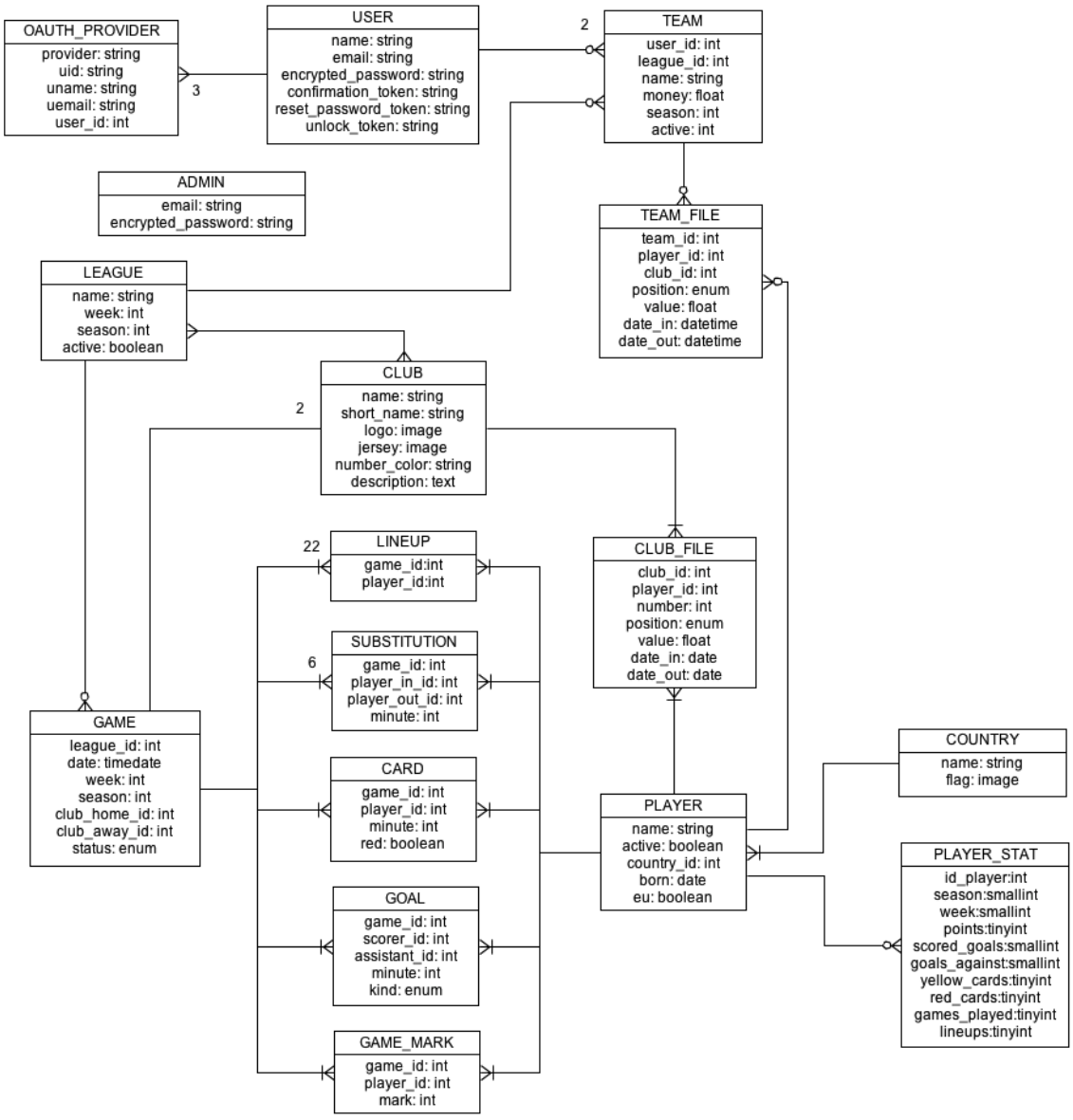

Figura 6.0.2: Estado actual de modelos.

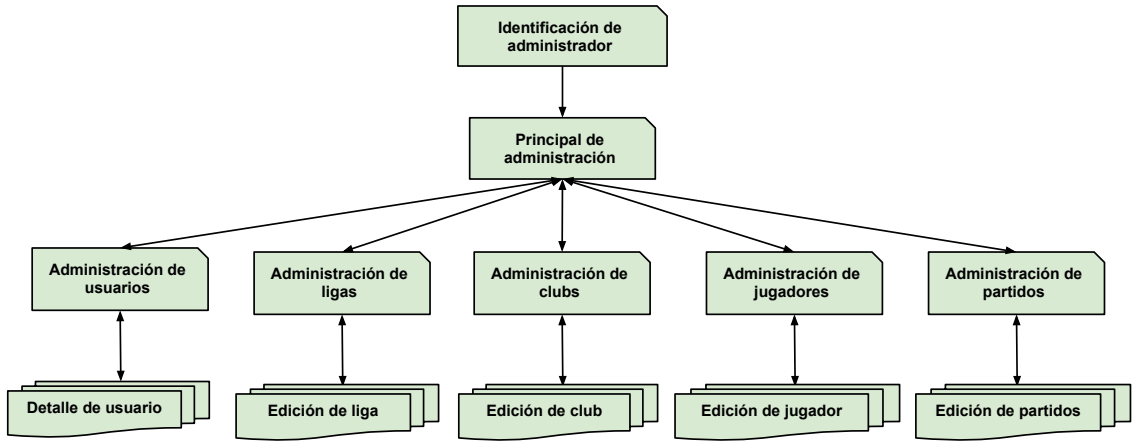

Figura 6.0.3: Estado actual de la navegación de administración.

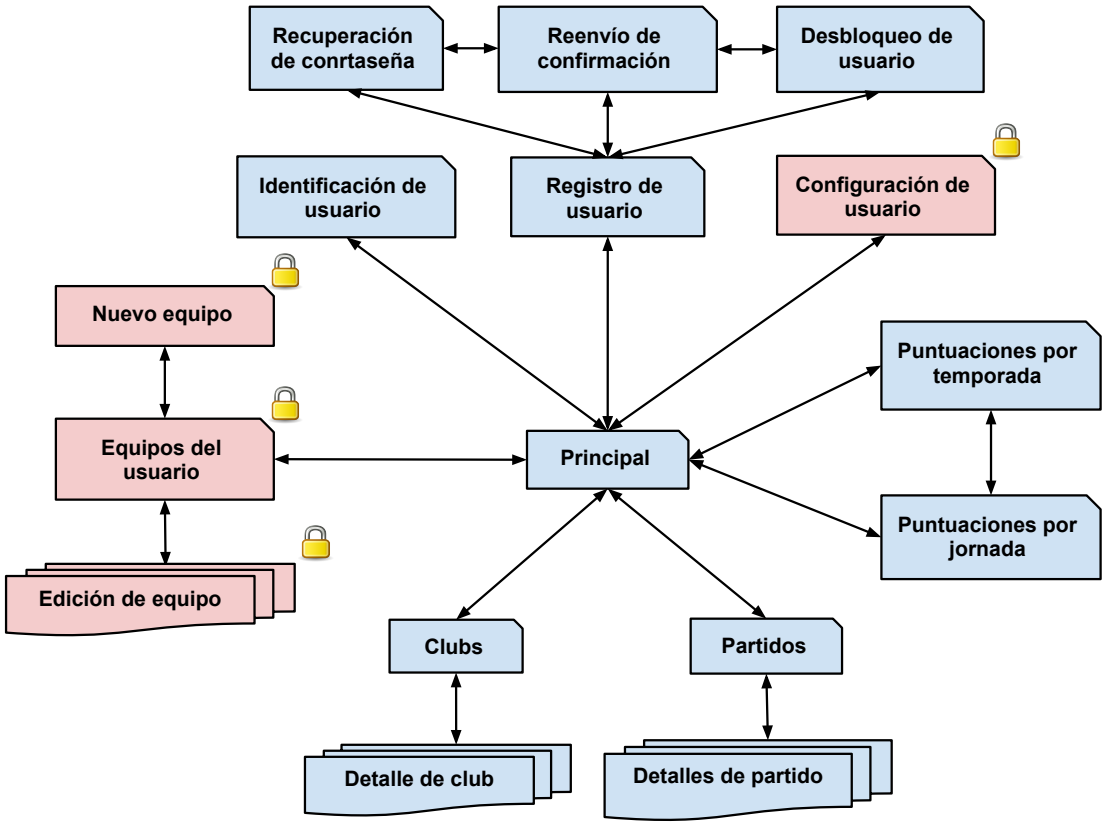

Figura 6.0.4: Estado actual de la navegación de usuario.

Capítulo 7

Ampliaciones

Durante el desarrollo del juego han ido surgiendo ideas y necesidades, que aun no siendo necesarias para alcanzar el estado actual de la aplicación, se plantean a continuación como posibles siguientes iteraciones a realizar.

7.1. Mercado de jugadores

El precio actual de los jugadores es un valor estático establecido en función de una valoración objetiva de la calidad de cada jugador. Debido a la gran cantidad de jugadores, su precio inicial puede no estar ajustado al nivel de juego actual. En muchos casos puede llegar a haber jugadores inicialmente muy baratos que estén obteniendo puntuaciones muy altas, y el juego quedaría desequilibrado, ya que los usuarios podrían contar con ese jugador sin problemas de presupuesto.

Para evitar esta situación, la siguiente iteración a implementar consistiría en crear un mercado de jugadores con valores autogestionados y más acordes con las puntuaciones del juego. Ya que actualmente se cuenta con una buena cantidad de datos sobre los jugadores, el precio inicial de los mismos debería establecerse a partir de las puntuaciones medias por partido. Además, al cierre de cada jornada, habría que calcular un icremento o decremento del valor del jugador en función de su nueva media.

Tal como está creado el juego, los precios de los jugadores en los equipos de los usuarios se almacenan en el momento de la compra, por lo que podemos plantear dos mecánicas de juego:

- Cuando un usuario vende un jugador, lo hace por el mismo precio al que lo compró. De esta forma se tendría un mercado estable y la variación en los precios afectaría únicamente a la dificultad a la hora de comprar buenos jugadores.
- Cuando un usuario vende un jugador, lo hace por el precio actual del jugador. En este caso, se estaría creando un mercado dinámico dentro del juego y además de dificultad de compra, añade más estrategia en las decisiones de venta.

Está claro que el segundo punto añade más alicientes al juego, pero crear un mercado dinámico que evoluciona solo, podría crear situaciones que estropeasen la experiencia de juego, como un equipo sin suficiente dinero para fichar jugadores o un usuario que, especulando, pueda llegar a comprar

muchos de los mejores jugadores del juego. Por esto antes de elegir una de las dos, sería recomendable realizar simulaciones con los datos almacenados hasta ese momento.

7.2. Incentivos/Recompensas

Otro concepto interesante a añadir en una futura iteración son las recompensas a los usuarios para motivar su uso de la aplicación. Algo muy extendido en los últimos tiempos en aplicaciones es la *gamification* (también conocida como gamificación o ludificación), que consiste en aplicar mecánicas de juegos en contextos ajenos a ellos. Algunos ya están presentes en la aplicación, como el uso de los puntos y las clasificaciones, pero otro que añadiría un gran aliciente al juego es el uso de logros o insignias. Esta mecánica consiste registrar ciertas acciones que realizan los usuarios de forma que puedan ser vistas por el resto de usuarios y premiadas con algún tipo de recompensa dentro del juego.

La plataforma social de videojuegos de *Microsoft*, *Xbox Live* con más de 10 años en activo, concede a sus usuarios a medida que progresan en sus juegos los denominados logros, una imagen y descripción de lo que han hecho, además de puntos de juego que más tarde pueden canjear por productos en la tienda de la plataforma.

Otro claro ejemplo en el mundo web es la aplicación *Foursquare*[1], una aplicación web social de registro de visitas en localizaciones reales, donde los usuarios van ganando insignias y rangos a medida que pasan por diferentes sitios.

Para el juego, la idea es definir las acciones que registren una serie de logros en el perfil del usuario, como por ejemplo, crear un equipo, el primer cambio de la jornada, hacer más de 50 puntos en una jornada o compartir nuestra puntuación en una red social. Además, asociar a cada una de ellas una cantidad de créditos como recompensa. Estos créditos serían una unidad interna del juego y distinta de los puntos de los equipos, que los usuarios podrían canjear por nuevas acciones dentro del juego, como conseguir un cambio extra, más presupuesto para el equipo o retar a otros usuarios.

Estos añadidos darían algo más de estrategia (con los créditos), fomentarían a los usuarios a usar la aplicación para conseguir logros y créditos y ayudarían a extender la aplicación en las redes sociales.

7.3. Modelo de negocio

La idea original siempre ha sido crear una aplicación accesible y compensada para todo el mundo. Si la aplicación crece es posible que se necesite ampliar los recursos, como por ejemplo mejores servidores si aumenta el número de usuarios o personas dedicadas al mantenimiento de datos si se intenta expandir a más ligas. En este caso, lo más recomendable sería buscar un modelo de negocio que no rompa con la igualdad dentro del juego, pero que pueda proporcionar mantenimiento para los nuevos gastos generados.

Como ya hemos visto en apartados anteriores, otros juegos monetizan completamente las acciones del juego u ofrecen cuentas de pago mensual para acceder a ciertas funcionalidades. Pero existen otros modelos de negocio aplicables a nuestro caso.

Añadir publicidad a la página es una opción viable. Si la aplicación consigue visitas, con una temática tan definida, no debería ser complicado conseguir publicidad de casas deportivas, apuestas

[1] Foursquare: `https://es.foursquare.com/`.

o similares. El aporte monetario iría en consecuencia con el uso de la aplicación, con lo que podrían compensarse los gastos necesarios. Añadir publicidad a la página puede estropear la experiencia de usuario al resultar molesta. Se considera que es mejor proporcionar un buen servicio y ofrecer buenas funcionalidades para mantener a los usuarios afines con el juego. Aun así, esta opción no debe descartarse.

La estrategia más adecuada en este caso son los micropagos, que como su propio nombre indica consiste en cobrar a los usuarios pequeñas cantidades de dinero por determinadas acciones dentro del juego. Es un modelo muy extendido actualmente en juegos a través de Internet y denominados *free-to-play*, en los que el juego se distribuye gratuitamente en la red y permite la compra de objetos dentro del juego para usar con los personajes, pero que no son necesarios para la experiencia normal de juego. *Valve*[2], una de las compañías más importantes en el sector del videojuego, lo ha puesto en práctica con algunos de sus juegos más conocidos como son el *Dota 2* o el *Team Fortress 2*. Éste último, por ejemplo, a multiplicado sus ingresos por doce desde que se volvió *free-to-play*[13].

Para aplicar este modelo en el juego, tras añadir un sistema de créditos como el comentado en el apartado anterior, bastaría con añadir un proceso de compra de créditos y añadir opciones para canjearlos, como la posibilidad de añadir un escudo personalizado a tu equipo o poder adquirir más equipos.

[2]Valve: `http://www.valvesoftware.com/`.

Capítulo 8

Conclusiones

En este apartado se muestran las principales conclusiones obtenidas a partir de las premisas planteadas al comienzo del proyecto.

El primero de los objetivos, referente a la aplicación de técnicas de desarrollo ágil, ha sido una de las principales motivaciones del proyecto como complemento a la formación académica, ya que cada vez son más utilizadas en el mundo Web. Su aplicación en este proyecto ha permitido un proceso de desarrollo muy dinámico y adaptable en los momentos en los que se han introducido cambios. El desarrollo dirigido por pruebas obliga al desarrollador a pensar en la implementación (cómo hacerlo) de funcionalidades (qué hacer) durante el planteamiento de las pruebas. Esto amplía a las metodologías tradicionales que, durante los procesos de definición, únicamente se centran en qué conseguir, sin pensar en cómo conseguirlo. Además, la aplicación de integración continua durante el desarrollo ha proporcionado una mayor seguridad en el proceso de despliegue de la aplicación. Esto es debido a que se debe superar la batería de pruebas en el momento de añadir nuevas funcionalidades, con lo que es poco probable desplegar una aplicación con problemas de funcionamiento importantes. Este factor constituye un aspecto fundamental para ofrecer un buen servicio en una aplicación web en constante funcionamiento.

En relación al segundo y tercer objetivos, referentes al desarrollo en iteraciones y a la disponibilidad de un entorno de producción, se ha conseguido el desarrollo de partes de la aplicación con funcionalidad propia, similares a un prototipo en constante evolución. Gracias a esto, y a tener la aplicación siempre desplegada en un entorno de producción accesible a través de Internet, los usuarios han podido reportar su experiencia para depurar y mejorar la aplicación a medida que ha ido creciendo. Además, este flujo constante de información vía Internet con usuarios directos, ha permitido suplir la carencia de un cliente con el que mantener reuniones periódicas.

En último lugar, y haciendo mención al uso de tecnologías ágiles y de código abierto como el caso de *Ruby on Rails*, se ha podido comprobar que su uso facilita en gran medida el desarrollo de aplicaciones y que no se requiere de una gran inversión económica. Mediante la utilización de una multitud de herramientas disponibles y creadas por la comunidad, se ha conseguido incrementar la productividad general del proceso. Así por ejemplo, reutilizando recursos que resuelven problemas comunes se ha conseguido mejorar la velocidad de desarrollo, y haciendo uso de herramientas como las descritas en esta memoria para mantener, perfeccionar y probar nuestra aplicación, se ha aumentando la calidad del producto final.

Además de estas conclusiones cabe destacar que la principal dificultad encontrada, y que ha supuesto un importante lastre en el desarrollo del proyecto, ha sido la dependencia de datos externos. Sin el acceso automatizado a una fuente de información fiable, se obliga al administrador del sistema a invertir una gran cantidad de tiempo en la búsqueda, cotejación e introducción de los datos necesarios para el funcionamiento de la aplicación.

En definitiva, *onLeague* es un ejemplo de juego de simulación vía web que aprovecha las virtudes de las metodologías de desarrollo ágil para crear una aplicación en un entorno de producción con usuarios reales.

Bibliografía

[1] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, oct. de 1999. ISBN: 978-0201616415.

[2] Kent Beck. *Test Driven Development: By Example*. Addison-Wesley Professional, nov. de 2002. ISBN: 978-0321146533.

[3] Mike Beedle y col. *Manifiesto para el Desarrollo Ágil*. 2001. URL: http://agilemanifesto.org/iso/es/principles.html.

[4] Steve Burbeck. «Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)». En: (1987). URL: http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html.

[5] Scott Chacon. *Git Internals*. PeepCode. URL: https://peepcode.com/products/git-internals-pdf.

[6] Scott Chacon. *Pro Git*. Apress, ago. de 2009. ISBN: 978-1430218333. URL: http://git-scm.com/book.

[7] Rails Community. *Ruby on Rails guides*. URL: http://guides.rubyonrails.org/.

[8] The jQuery Foundation. *jQuery Api*. URL: http://api.jquery.com/.

[9] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2003. ISBN: 978-0321127426.

[10] Hillel Glazer y col. *CMMI or Agile: Why Not Embrace Both!* Inf. téc. 2008. URL: http://www.sei.cmu.edu/library/abstracts/reports/08tn003.cfm.

[11] Andrew Hunt y David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, oct. de 1999. ISBN: 978-0201616224.

[12] MARCA. *Gía de la liga 2013*. Grupo Unidad Editorial, ago. de 2013.

[13] Patrick Miller. *GDC 2012: How Valve made Team Fortress 2 free-to-play*. Mar. de 2012. URL: http://www.gamasutra.com/view/news/164922/GDC_2012_How_Valve_made_Team_Fortress_2_freetoplay.php.

[14] Aaron Sumner. *Everyday Rails Testing with RSpec*. Leanpub, 2012. URL: https://leanpub.com/everydayrailsrspec.

[15] Twitter. *Twitter Bootstrap guides*. URL: http://twitter.github.com/bootstrap/getting-started.html.

Apéndices

Apéndice A

Guía de usuario

Este documento explica como acceder a la aplicación en producción y ofrece una guía rápida de las principales funcionalidades.

A.1. Datos

Puede accederse a la aplicación desde un navegador web (es recomendado usar *Chrome*, *Firefox* u *Opera*) con los siguientes datos:

Dirección: `http:\www.onleague.org`

Usuario: shigeru

Contraseña: miyamoto

Este usuario y contraseña son iguales para todo el mundo y sólo es necesario introducirlos la primera vez que se accede por sesión. Si en el momento de acceder no son solicitados, significa que la página ya está accesible para todo el mundo.

Además puede accederse desde la red a los siguientes recursos:

Código fuente: `https://github.com/rsierra/onLeague`

Memoria: `http://rsierra.github.com/onLeague/doc/onleague.pdf`

Presentación: `http://rsierra.github.com/onLeague/slides/`

A.2. Registro

Una vez dentro de la página puede navegarse libremente por los partidos y clubs de la liga y por las clasificaciones de equipos. Para acceder a la sección de equipos se requiere registrar un usuario. Para hacerlo es necesario seguir el enlace de la parte superior derecha **Identificarse**. Este enlace despliega una ventana donde introducir nuestro nombre de usuario y contraseña si ya estamos registrados o realizar un nuevo registro de dos formas posibles:

Registro convencional: desde la ventana de identificación, usar el botón **Registrarse** para llegar al formulario de registro convencional. Proporcionar los datos solicitados y finalmente confirmar el registro desde el enlace proporcionado en el correo electrónico recibido en la dirección indicada en el registro.

Registro con proveedor: el método más sencillo para registrarse es usando alguno de los proveedores disponibles. Si ya tienes usuario en alguno de los proveedores, pulsa el icono del deseado. Serás redirigido a la página del proveedor en cuestión donde se deben aceptar los permisos para onLeague. Después volverás a la página identificado. En el caso de *Twitter* en la vuelta deberás proporcionar un correo electrónico, ya que es necesario para el registro y *Twitter* no lo proporciona.

En el caso de haber creado el usuario por el método convencional, una vez identificados, podemos igualmente agregar proveedores desde la configuración de usuario, desde el icono de engranaje en la esquina superior derecha.

Si se tiene asociado un proveedor al usuario, puede usarse el mismo icono del proveedor para identificarse rápidamente en la aplicación.Si el usuario ha sido creado únicamente con proveedor y posteriormente quiere acceder a la aplicación con usuario y contraseña, debe primero solicitar un cambio de contraseña desde el botón **¿Olvidaste tu contraseña?** de la ventana de identificación, ya que esta nunca ha sido establecida.

A.3. Creación de equipo

Para crear un equipos es necesario estar identificado en la aplicación con un usuario. Desde la sección de **Tus equipos** se puede crear un nuevo equipo y darle el nombre deseado.

Una vez creado, se ha de acceder a la edición del equipo seleccionándolo desde la misma sección de **Tus equipos** y es necesario activarlo para empezar a acumular puntos. Para activarlo basta con comprar jugadores cumpliendo las restricciones indicadas. Cuando el equipo esté listo para activarse se indicará automáticamente. Una vez el equipo esté activo, sólo se pueden realizar 3 cambios por jornada.

Toda la edición del equipo se realiza desde esta misma página. La búsqueda de jugadores se actualiza en la propia sección, estando ordenados por mayor puntuación y el precio más barato. Desde esta lista pueden comprarse jugadores, y desde la alineación del equipo pueden venderse los ya adquiridos. En el caso de no cumplirse alguna restricción de compra o venta, el sistema no permite la operación y muestra un mensaje describiendo el problema. Cuando el equipo ya está activo y solo se disponen de 3 cambios por jornada, tanto las compras como las ventas, se acumulan sin hacerse efectivas. De esta forma puede comprobarse en que estado quedará el equipo con esas negociaciones, para finalmente confirmar o rechazar los cambios.