Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

Aplicación web Java para gestionar ligas de futbol profesionales y competiciones de equipos virtuales

Estudante: Pablo Villamarín Rodríguez
Dirección: José Losada Pérez

A Coruña, xuño de 2021.

A mi familia y amigos

Agradecimientos

A mi familia, amigos, compañeros y profesores que en algún momento me ayudaron y me empujaron a llegar hasta aqui. A mi tutor Jose por ayudarme en todo momento tanto en el TFG como cuando fui alumno suyo en sus asignaturas.

Resumen

Este proyecto consistirá en el análisis, diseño e implementación de una aplicación web que permitirá gestionar varias ligas de fútbol profesional. Además, los usuarios de la aplicación podrán participar en una competición que consiste en elegir un equipo virtual con sus futbolistas preferidos de entre todos los jugadores que forman parte de algún equipo de esas ligas.

La aplicación permitirá gestionar todos los datos referentes a las ligas de fútbol, incluyendo toda la información relativa a los equipos, a los jugadores y a los diferentes partidos. Jornada a jornada, se irá introduciendo una puntuación a cada futbolista que haya disputado el partido y en base a esta puntuación, se creará un ranking entre todos los equipos virtuales de los usuarios de la web.

Esta aplicación se ha diseñado siguiendo los principios de la arquitectura MVC y las principales tecnologías utilizadas para desarrollar esta aplicación son Hibernate, Spring y Thymeleaf.

La metodología empleada en este proyecto es una de las metodologías ágiles más utilizadas denominada Scrum, que se caracteriza por ser iterativa e incremental. En cada iteración o sprint se abarca un subconjunto del total de funcionalidades a implementar. Como resultado de cada iteración se obtiene un software funcional.

Finalmente, la aplicación se integrará con un servicio web real, se utilizará Google como servicio de autenticación externo.

Abstract

This project will consist of the analysis, design and implementation of a web application that will allow the management of several professional football leagues. In addition, the users of the application will be able to participate in a competition that consists of choosing a virtual team with their favourite football players from among all the players that form part of a team in these leagues.

The application will make it possible to manage all the data relating to the football leagues, including all the information relating to the teams, the players and the different matches. On each match day, a score will be introduced for each player who has played the match and based on this score, a ranking will be created among all the virtual teams of the users of the web.

This application has been designed following MVC architecture principles and the main technologies used to develop this application are Hibernate, Spring and Thymeleaf.

The methodology used in this project is one of the most widely used agile methodologies called Scrum, which is characterised by being iterative and incremental. In each iteration or sprint, a subset of the total functionalities to be implemented is covered. As a result of each iteration a functional software is obtained.

Finally, the application will be integrated with a real web service, Google will be used as an external authentication service.

Palabras clave:

- Aplicación web
- Hibernate
- Spring
- Scrum
- Sprint
- Thymeleaf
- Liga
- Fútbol
- Usuario
- Torneo

Keywords:

- Web application
- Hibernate
- Spring
- Scrum
- Sprint
- Thymeleaf
- League
- Football
- User
- Tournament

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introdución

Este trabajo consistirá en el análisis, diseño e implementación de una aplicación web que permitirá gestionar varias ligas de fútbol profesional. Además, los usuarios de la aplicación podrán participar en una competición que consiste en elegir un equipo virtual con sus futbolistas preferidos de entre todos los jugadores que forman parte de algún equipo de esas ligas.

Hay dos tipos de usuarios: los administradores y los usuarios que harán uso de sus servicios. Según el tipo de usuario que haga uso de la aplicación, tendrá acceso a distintas funcionalidades. La aplicación permite que un usuario se registre, inicie y cierre sesión en la página, modifique los datos de su perfil y cambie su contraseña. Cualquier usuario, aunque no esté registrado en la aplicación, podrá buscar información sobre las ligas, clubes y jugadores. Los usuarios registrados, además, podrán acceder a una competición en la que elegirán a 11 jugadores de la liga a la que corresponda el torneo y competiran contra otros usuarios hasta el final de la liga. Los administradores podrán realizar tareas relacionadas con la gestión de las ligas, clubes, jugadores y la competición entre los usuarios.

Para el desarrollo de la aplicación se ha empleado una metodología ágil, iterativa e incremental denominada Scrum. Por lo tanto, el proyecto se ha dividido en una serie de iteraciones y se han implementado una o varias funcionalidades en cada una de estas iteraciones. El diseño de la aplicación se ha hecho siguiendo el patrón de arquitectura MVC. El modelo es la capa donde se trabaja con los datos, por tanto contendrá mecanismos para acceder a la información y también para actualizar su estado. La vista contiene el código de nuestra aplicación que va a producir la visualización de las interfaces de usuario, o sea, el código que nos permitirá renderizar los estados de nuestra aplicación en HTML. El controlador es una capa que sirve de enlace entre la vista y el modelo, respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra aplicación. Para implementar la aplicación se han usado varias herramientas: Spring, Hibernate y Thymeleaf son las más importantes.

Capítulo 2

Fundamentos tecnológicos

El presente capítulo tiene como objetivo ofrecer una idea general de cada una de las herramientas o tecnologías analizadas y/o usadas en el desarrollo de este proyecto.

2.1 Hardware

Se ha utilizado un ordenador personal con las siguientes características:

- Sistema operativo: Windows 10
- Procesador: Intel Core i7-6700 HQ 2.60 GHz
- Memoria RAM: 8GB
- Tipo de sistema: 64 bits

2.2 Lenguajes

2.2.1 Java

Es un lenguaje [1] basado en clases y orientado a objetos. Este lenguaje se ha elegido principalmente por el conocimiento y familiaridad del mismo. Con este lenguaje se desarrollará la parte del modelo y controlador del proyecto.

2.2.2 JavaScript

JavaScript [2] es un lenguaje de programación ligero, interpretado, o compilado justo-a-tiempo (just-in-time). Si bien es más conocido como un lenguaje de scripting (secuencias de comandos) para páginas web, y es usado en muchos entornos fuera del navegador, tal como Node.js, Apache CouchDB y Adobe Acrobat. JavaScript es un lenguaje de programación basada en prototipos, multiparadigma, de un solo hilo, dinámico, con soporte para programación

orientada a objetos, imperativa y declarativa. Este lenguaje nos ayudará con la parte de la vista de la aplicación.

2.2.3 HTML

Hace referencia al lenguaje de marcado para la elaboración de páginas web, permite modificar la apariencia con ayuda de CSS y JavaScript. Se ha elegido este lenguaje para desarrollar la vista de la aplicación.

2.2.4 CSS

Las Hojas de estilo en cascada (del ingles Cascading Stylesheets CSS) se usan para darle estilo y posicionar visualmente el contenido escrito en HTML. CSS se puede usar, por ejemplo, para cambiar la fuente, el color, el tamaño y el espaciado del contenido, para formar múltiples columnas, añadir animaciones y otros elementos decorativos.

2.2.5 SQL

SQL es un lenguaje de dominio específico utilizado en programación, diseñado para administrar, y recuperar información de sistemas de gestión de bases de datos relacionales. Se utiliza en el modelo de la aplicación para la gestión de los datos.

2.3 Frameworks y librerías

2.3.1 Hibernate

Hibernate [3] es una herramienta de mapeo objeto-relacional (ORM) para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los contenedores (beans) de las entidades que permiten establecer estas relaciones. Como todas las herramientas de su tipo, Hibernate busca solucionar el problema de la diferencia entre los dos modelos de datos coexistentes en una aplicación: el usado en la memoria de la computadora (orientación a objetos) y el usado en las bases de datos (modelo relacional). Para lograr esto permite al desarrollador detallar cómo es su modelo de datos, qué relaciones existen y qué forma tienen. Con esta información Hibernate le permite a la aplicación manipular los datos en la base de datos operando sobre objetos, con todas las características de la POO. Ofrece también un lenguaje de consulta de datos llamado HQL (Hibernate Query Language), al mismo tiempo que una API para construir las consultas programáticamente (conocida como ”criteria”).

2.3.2 Spring

Spring [4] es un framework para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto para la plataforma Java. Spring nos permite desarrollar aplicaciones de manera más rápida, eficaz y corta, saltándonos tareas repetitivas y ahorrándonos lineas de código.

2.3.3 Thymeleaf

Thymeleaf [5] es una librería Java que implementa un motor de plantillas que puede ser utilizado tanto en modo web como en otros entornos no web. Se acopla muy bien para trabajar en la vista del patrón MVC de aplicaciones web, pero puede procesar cualquier archivo XML, incluso en entornos desconectados.

Proporciona un módulo opcional para la integración con Spring MVC, por lo que se puede utilizar para reemplazar completamente a los archivos JSP en tus aplicaciones construidas con esta tecnología.

El objetivo principal de Thymeleaf es permitir la creación de plantillas de una manera elegante y un código bien formateado. Sus dialectos Standard y SpringStandard permiten crear plantillas que se pueden visualizar correctamente en los navegadores de Internet, por lo que también funcionan como prototipos estáticos.

2.4 Herramientas para el desarrollo

2.4.1 Eclipse

Eclipse [6] es una plataforma de desarrollo, diseñada para ser extendida a través de plug-ins. Fue concebida desde sus orígenes para convertirse en una plataforma de integración de herramientas de desarrollo. No tiene en mente un lenguaje específico, sino que es un IDE genérico, aunque goza de mucha popularidad entre la comunidad de desarrolladores del lenguaje Java usando el plug-in JDT que viene incluido en la distribución estándar del IDE.

Proporciona herramientas para la gestión de espacios de trabajo, escribir, desplegar, ejecutar y depurar aplicaciones.

2.4.2 Maven

Maven [7] es una herramienta de software para la gestión y construcción de proyectos Java. Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

2.4.3 Git

Es un software de control de versiones pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.

2.4.4 Derby

Apache Derby es un sistema gestor de base de datos relacional escrito en Java que puede ser empotrado en aplicaciones Java y utilizado para procesos de transacciones en línea. Inicialmente distribuido como IBM Cloudscape, Apache Derby es un proyecto open source disponible en virtud de la Apache 2.0 License.

Capítulo 3

Metodología

PARA el desarrollo del proyecto se ha decidido utilizar una metodología ágil denominada Scrum. A continuación se muestran las principales características de una metodología ágil, Scrum y la adaptación de esta metodología al proyecto.

3.1 Metodología ágil

Las metodologías ágiles son aquellas que permiten adaptar la forma de trabajo a las condiciones del proyecto, consiguiendo flexibilidad e inmediatez en la respuesta para amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno. A continuación enumeramos algunas de las ventajas que nos brinda la gestión ágil de proyectos:

- **Mejora de la calidad del producto:** Estas metodologías fomentan el enfoque proactivo de los miembros del equipo en la búsqueda de la excelencia del producto. Además, la integración, comprobación y mejora continúa de las propiedades del producto mejora considerablemente el resultado final.
- **Mayor satisfacción del cliente:** El cliente está más satisfecho al verse involucrado y comprometido a lo largo de todo el proceso de desarrollo. Mediante varias demostraciones y entregas, el cliente ve en tiempo real las mejoras introducidas en el proceso.
- **Mayor motivación de los trabajadores:** Los equipos de trabajo autogestionados, facilitan el desarrollo de la capacidad creativa y de innovación entre sus miembros.
- **Trabajo colaborativo:** La división del trabajo en distintos equipos y roles junto al desarrollo de reuniones frecuentes, permite una mejor organización del proyecto.
- **Uso de métricas más relevantes:** Las métricas utilizadas para estimar parámetros como tiempo, coste, rendimiento, etc. son normalmente más reales en proyectos ágiles

que en los tradicionales. Gracias a la división en pequeños equipos y fases se puede ser más conscientes de lo que está sucediendo.

- **Mayor control y capacidad de predicción:** La oportunidad de revisar y adaptar el producto a lo largo del proceso ágil, permite a todos los miembros del proyecto ejercer un mayor control sobre su trabajo, cosa que permite mejorar la capacidad de predicción en tiempo y costes.

- **Reducción de costes:** La gestión ágil del proyecto elimina prácticamente la posibilidad de fracaso absoluto en el proyecto, porque los errores se van identificando a lo largo del desarrollo en lugar de esperar a que el producto esté acabado y toda la inversión realizada

3.2 Scrum

Se caracteriza por ser la «metodología del caos» que se basa en una estructura de desarrollo incremental, esto es, cualquier ciclo de desarrollo del producto y/o servicio se desgrana en «pequeños proyectos» divididos en distintas etapas: análisis, desarrollo y testing. En la etapa de desarrollo se encuentra lo que se conoce como interacciones del proceso o Sprint. Un Sprint es un periodo de tiempo fijo, definido por el equipo, pero preferentemente corto (normalmente de 2 a 4 semanas), en el cual se realiza el trabajo. En cada Sprint, el equipo de desarrollo debe construir y entregar un Incremento de Producto, una entrega parcial que consiste en una versión del producto realizada durante el proceso de desarrollo. Los incrementos deben ser funcionales, estar operativos y deben mostrar una mejoría respecto del anterior.

Otra de las claves de Scrum son las historias de usuario. Una historia de usuario es una explicación general e informal de una función de software escrita desde la perspectiva del usuario final. Su propósito es articular cómo un elemento de trabajo entregará un valor particular al cliente.. Las historias de usuario suelen expresarse con una frase simple con la siguiente estructura:

Como [rol], [quiero] [funcionalidad]

Las historias de usuario forman el Product Backlog que es una lista de características que han sido priorizadas, y contiene descripciones breves sobre todo lo que se desea para el producto que se va a desarrollar.

3.3 Adaptación de la metodología al proyecto

En Scrum existen varios roles: Product Owner es el encargado del proyecto, este rol será asumido por el profesor, Scrum master: cuya función es eliminar los trabas que obstaculizan

alcanzar los objetivos y equipo de desarrollo: es el responsable de la entrega del producto. Está compuesto por un grupo de profesionales multidisciplinar capacitado para realizar el proyecto, serán desempeñados por el alumno. En un proyecto real todos estos roles están formado por unas 6-7 personas pero aquí no procede, al igual que habría un cliente que es al que se le presenta el producto final, pero como no existe un cliente en este proyecto lo asumirá el profesor. La duración de los sprints suele ser entre 2-4 semanas, en este proyecto puede que en algún caso se haya sobrepasado debido a que muchos de los contenidos de la aplicación eran nuevos para el alumno y al tiempo de implementación habría que sumarle el tiempo de aprendizaje. A medida que se ha ido desarrollando el proyecto se ha ido reduciendo el tiempo de los sprints debido a que el alumno ha ido familiarizándose con el entorno de trabajo.

Capítulo 4

Analisis

4.1 Historias de usuario

Como se ha dicho anteriormente, las historias de usuario son una parte muy importante de la metodología Scrum, a continuación se mostrará en la tabla 4.1 (Pag. 12) las historias de usuario que se definieron al principio del proyecto.

4.2 Sprints

El proyecto se ha dividido en 6 sprints. Los criterios seguidos para realizar la agrupación de historias de usuario en los distintos sprints han sido:

- Implementar primero las funcionalidades esenciales de la aplicación, y aquéllas de las que dependan otras.
- Realizar primero las funcionalidades más sencillas. De esta forma, a medida que se adquiere familiaridad con la aplicación, se abarcarán funcionalidades más complejas con mayor eficacia
- Realizar en un mismo sprint funcionalidades relacionadas.
- Si varias funcionalidades relacionadas son demasiado complejas para realizarlas en un mismo sprint, se realizarán en sprints consecutivos.

A continuación se mostrarán todos los sprints con las historias de usuario asociadas a cada uno.

4.2.1 Sprint 0

En este sprint se implementarán las historias de usuario 1,2,3,4. El objetivo de este primer sprint es crear el esqueleto de la aplicación y empezar a entender como funciona el patrón

Id	Historias de usuario
1	Como cliente, quiero registrarme en la aplicación.
2	Como cliente o administrador, quiero iniciar sesión en la aplicación.
3	Como cliente o administrador, quiero actualizar mi perfil.
4	Como cliente o administrador, quiero cambiar mi contraseña.
5	Como cliente, quiero consultar una lista con las diferentes ligas y la tabla de clasificación de una liga determinada
6	Como administrador, quiero modificar los datos de una liga.
7	Como cliente, quiero consultar los datos de un equipo de futbol.
8	Como administrador, quiero modificar los datos de un equipo.
9	Como cliente, quiero consultar los datos de un jugador determinado.
10	Como administrador, quiero modificar los datos de un jugador.
11	Como administrador, quiero crear o modificar una jornada.
12	Como cliente, quiero consultar los resultados de una jornada.
13	Como administrador, quiero puntuar a los jugadores en cada partido.
14	Como administrador, quiero crear y modificar los datos de un partido.
15	Como cliente, quiero unirme a un torneo y elegir a los 11 jugadores con los que voy a competir con el resto de los usuarios.
16	Como cliente, quiero ver el equipo formado por los 11 jugadores que elegí y ver sus puntuaciones.
17	Como cliente, quiero saber los puntos que tengo en relación con los demás usuarios.
18	Como cliente, quiero ver toda la información relacionada con mi equipo favorito.
19	Como cliente, quiero buscar a un jugador por nombre o apellido.
20	Como cliente, quiero poder iniciar sesión con Google.

Tabla 4.1: Historias de usuario

MVC, Spring y el motor de plantillas Thymeleaf. Cuando se termine este sprint tanto el admin como un usuario cualquiera podría iniciar sesión, cerrar sesión, registrarse (solo el usuario), modificar su contraseña o cualquier información relacionada con su cuenta.

- Como cliente, quiero registrarme en la aplicación.
- Como cliente o administrador, quiero iniciar sesión en la aplicación.
- Como cliente o administrador, quiero actualizar mi perfil.
- Como cliente o administrador, quiero cambiar mi contraseña.

4.2.2 Sprint 1

En este sprint se implementarán las historias de usuario 5,6,7,8. El objetivo de este sprint es implementar las operaciones CRUD de las entidades League y Club, también se muestran los equipos de una liga en una tabla clasificatoria para que la pueda ver el usuario y la información de cada liga y club.

- Como cliente, quiero consultar una lista con las diferentes ligas y la tabla de clasificación de una liga determinada.
- Como administrador, quiero modificar los datos de una liga.
- Como cliente, quiero consultar los datos de un equipo de fútbol.
- Como administrador, quiero modificar los datos de un equipo.

4.2.3 Sprint 2

En este sprint se implementarán las historias de usuario 9,10,11,12. El objetivo de este sprint es implementar las operaciones CRUD de las entidades Player y LeagueProgram y el usuario puede ver una lista con todos los jugadores y una lista de todas las jornadas de una liga.

- Como cliente, quiero consultar los datos de un jugador determinado.
- Como administrador, quiero modificar los datos de un jugador.
- Como administrador, quiero crear o modificar una jornada.
- Como cliente, quiero consultar los resultados de una jornada.

4.2.4 Sprint 3

En este sprint se implementarán las historias de usuario 13,14. El objetivo de este sprint es implementar los partidos en cada uno habrá distintos eventos(gol, asistencia, tarjeta amarilla, tarjeta roja y sustituciones), al final del mismo se otorgarán unas puntuaciones con respecto a la actuación de cada jugador.

- Como administrador, quiero puntuar a los jugadores en cada partido.
- Como administrador, quiero crear y modificar los datos de un partido.

4.2.5 Sprint 4

En este sprint se implementarán las historias de usuario 15,16,17. El objetivo de este sprint es implementar la funcionalidad del torneo, una competición entre los usuarios registrados.

- Como cliente, quiero unirme a un torneo y elegir a los 11 jugadores con los que voy a competir con el resto de los usuarios.
- Como cliente, quiero ver el equipo formado por los 11 jugadores que elegí y ver sus puntuaciones.
- Como cliente, quiero saber los puntos que tengo en relación con los demás usuarios.

4.2.6 Sprint 5

En este ultimo sprint se implementarán las historias de usuario 18,19. El objetivo de este sprint es implementar la funcionalidad de que cada usuario pueda elegir un equipo favorito entre todos los registrados en la web y un buscador para encontrar mejor a los jugadores.

- Como cliente, quiero ver toda la información relacionada con mi equipo favorito.
- Como cliente, quiero buscar a un jugador por nombre o apellido.
- Como cliente, quiero poder iniciar sesión con Google.

Capítulo 5

Diseño

5.1 Arquitectura de la aplicación

La arquitectura de la aplicación corresponde a una aplicación web implementada en el lenguaje Java junto con las tecnologías Hibernate, JPA, Spring y utilizando el patrón MVC. Por tanto el proyecto estará estructurado según este patrón dividiéndolo en 3 subsistemas.

- **Modelo**: Es la representación de la información con la que el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la vista aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al modelo a través del controlador.

  Está formado por:

  - **Capa lógica de negocios**: está implementada utilizando Hibernate y JPA, en esta capa se encuentran las entidades y cada entidad tiene asociado una clase repository que es una API de bajo nivel para trabajar con las principales entidades persistentes de la aplicación (creación, actualización, borrado, etc.).
  - **Capa de servicio**: aquí se encuentran implementados todos los casos de uso en las diferentes clases.

- **Controlador**: es el encargado de gestionar las peticiones HTTP que llegan desde los clientes web, realizar las llamadas a la lógica de negocio y finalmente devolver los resultados en formato HTML. Se puede decir que es el intermediario entre la vista y el modelo.

- **Vista**: presenta el modelo (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario), por tanto requiere de dicho modelo la información que debe representar como salida.

En la figura 5.1 se muestra la arquitectura de la aplicación.

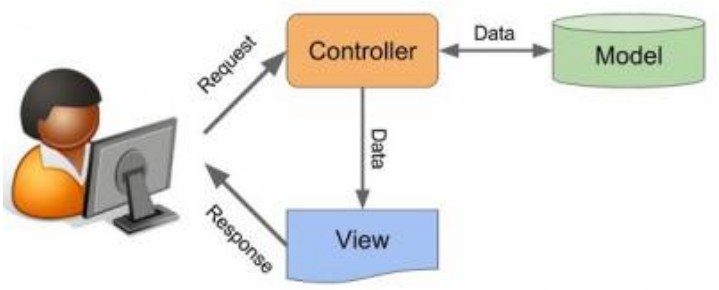

Figura 5.1: Arquitectura de la aplicación

5.2 Modelo

En el modelo se pueden diferenciar dos capas: la lógica de negocio, donde encontramos las entidades y las clases repository que son las clases donde están las funciones para acceder a los datos y la capa de servicio, donde están implementadas todas las funcionalidades de la aplicación.

5.2.1 Lógica de negocio

En el diagrama de la figura 5.2 se encuentran las entidades persistentes de la aplicación con todas las relaciones entre sí, y es en lo que se ha basado a la hora de implementar todas las funcionalidades de la aplicación. A continuación se detallan las entidades del modelo:

- User: Representa un usuario de la aplicación, puede ser un admin o usuario.
- Player: Representa un jugador de fútbol.
- Position: Representa una posición de un jugador de fútbol.
- Club: Representa un equipo de fútbol formado por jugadores.
- Formation: Representa un formación de un equipo de fútbol.
- League: Representa una liga formada por equipos de fútbol
- LeagueProgram: Representa cada una de las jornadas de una liga y está formada por partidos.

- Match: Representa partidos de fútbol en una jornada determinada.
- Event: Representa los eventos existentes en un partido de fútbol.
- Statistics: Representa las estadísticas de un jugador en un partido determinado.
- Tournament: Representa la competición interna entre usuarios de la aplicación.

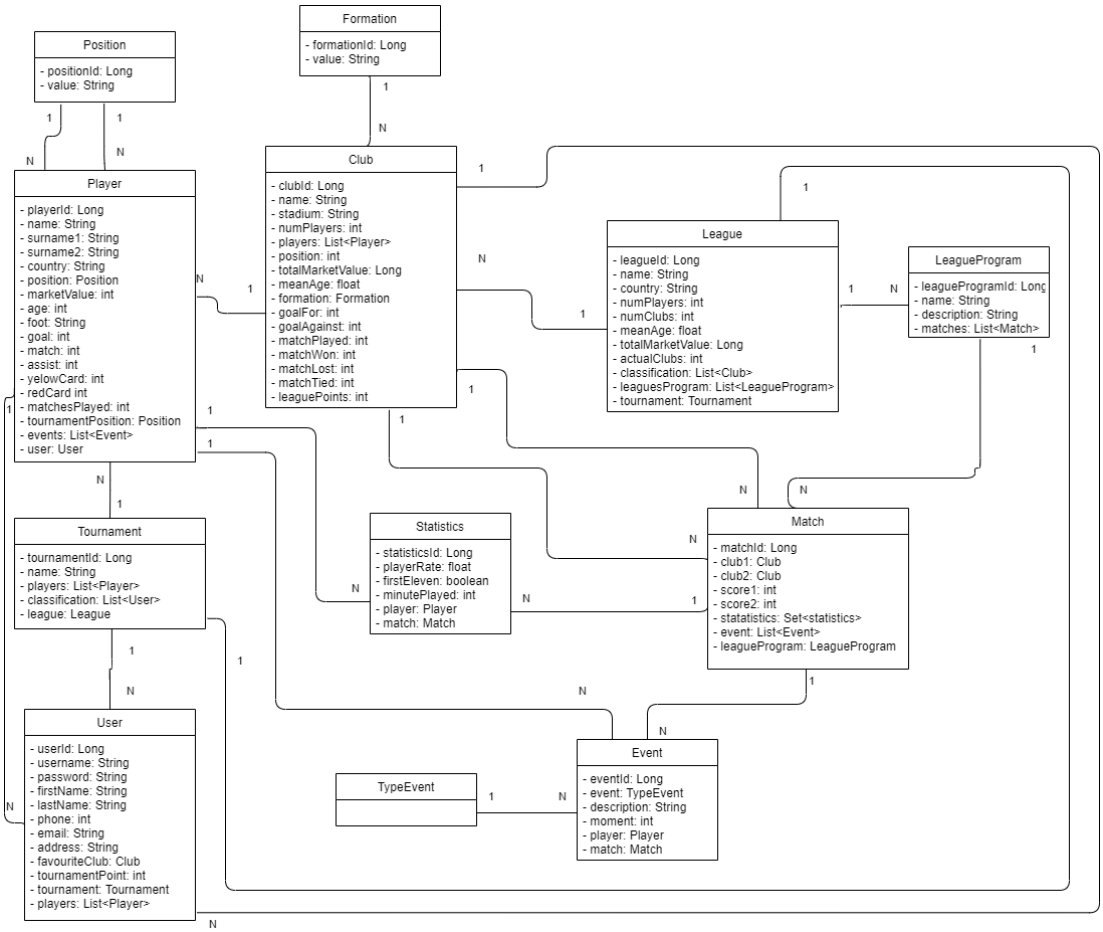

Figura 5.2: Diagrama de entidades

Cada entidad tiene asociada una clase Repository, a su vez cada clase extiende de la clase AbastractRepository, en esta clase están implementadas todas las funciones CRUD (create, read, update, delete) y algunas funciones de búsqueda. En cada una de las clases que extienden de AbastractRepository hay implementadas otras funciones necesarias para desarrollar las funcionales de la aplicación. A continuación mostraré un ejemplo de la entidad User como se muestra en la figura 5.3.

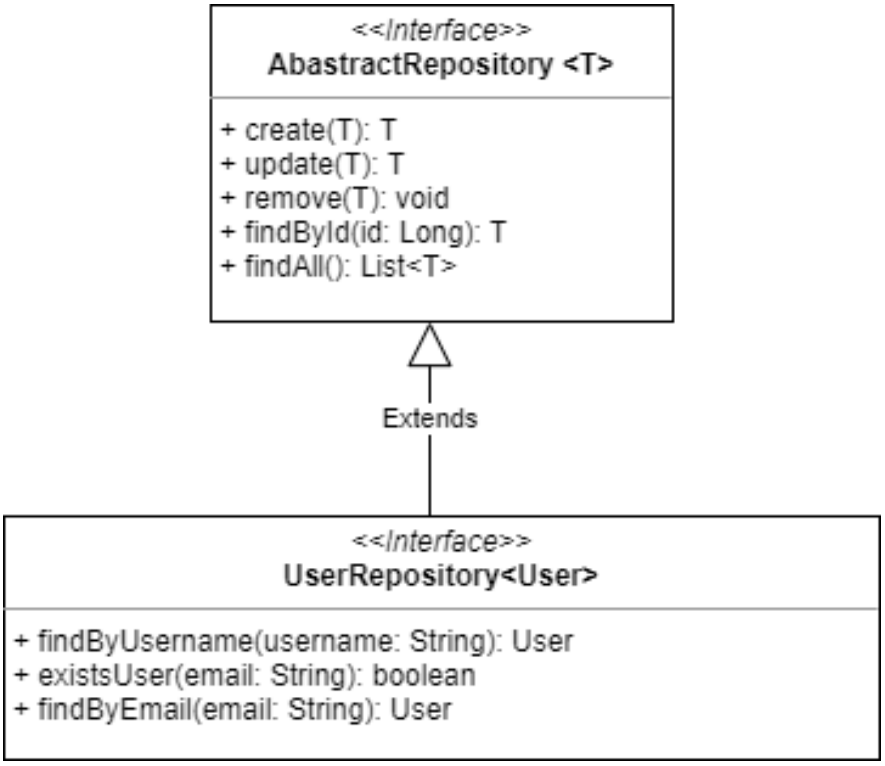

Figura 5.3: Diagrama de UserRepository

5.2.2 Servicio

En esta capa se agrupan las funcionalidades de la aplicación, se han dividido en 6 clases distintas.A continuación se mostrarán los diagramas de todos los servicios con sus respectivas funciones y una pequeña explicación de cada una de las clases que representan el servicio:

- UserService: En esta clase se incluyen las funcionalidades relacionadas con los usuarios como el login, cambio de contraseña, añadir un nuevo usuario, eliminar un usuario existente, actualizar los datos del perfil y buscar por email.

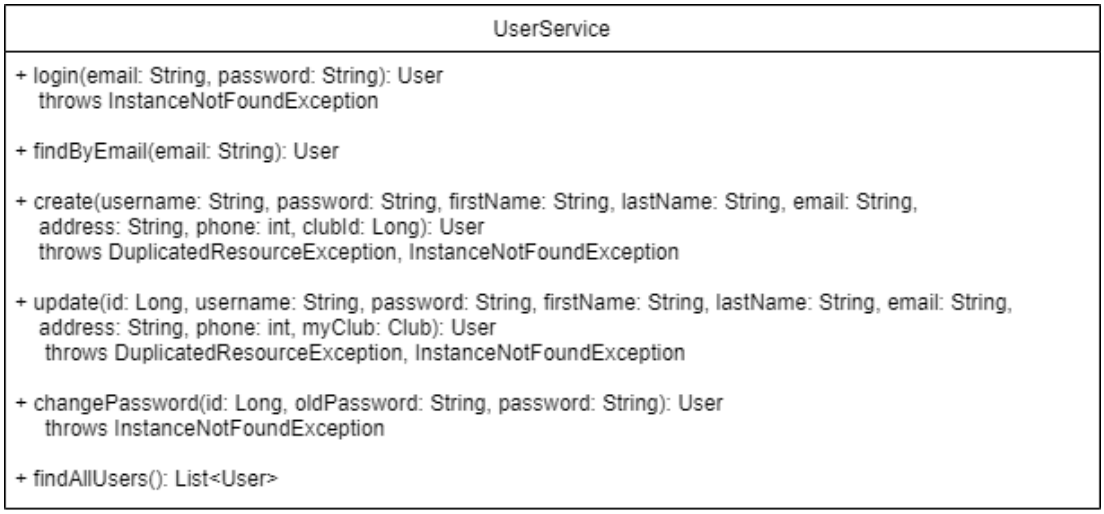

Figura 5.4: Diagrama de UserService

- PlayerService: En esta clase se incluyen las funcionalidades relacionadas con los jugadores y la posición como buscar por id, añadir un jugador, actualizar algún dato del jugador y eliminar un jugador existente.

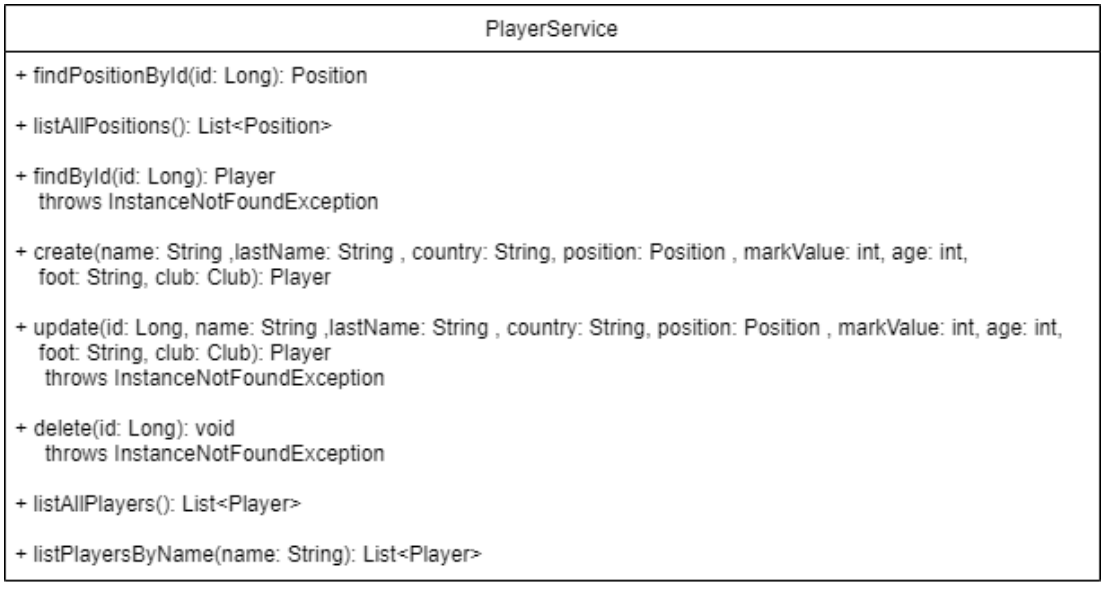

Figura 5.5: Diagrama de PlayerService

- ClubService: En esta clase se incluyen las funcionalidades relacionadas con los clubes y la formación como buscar por id, añadir un equipo, actualizarlo o eliminarlo.

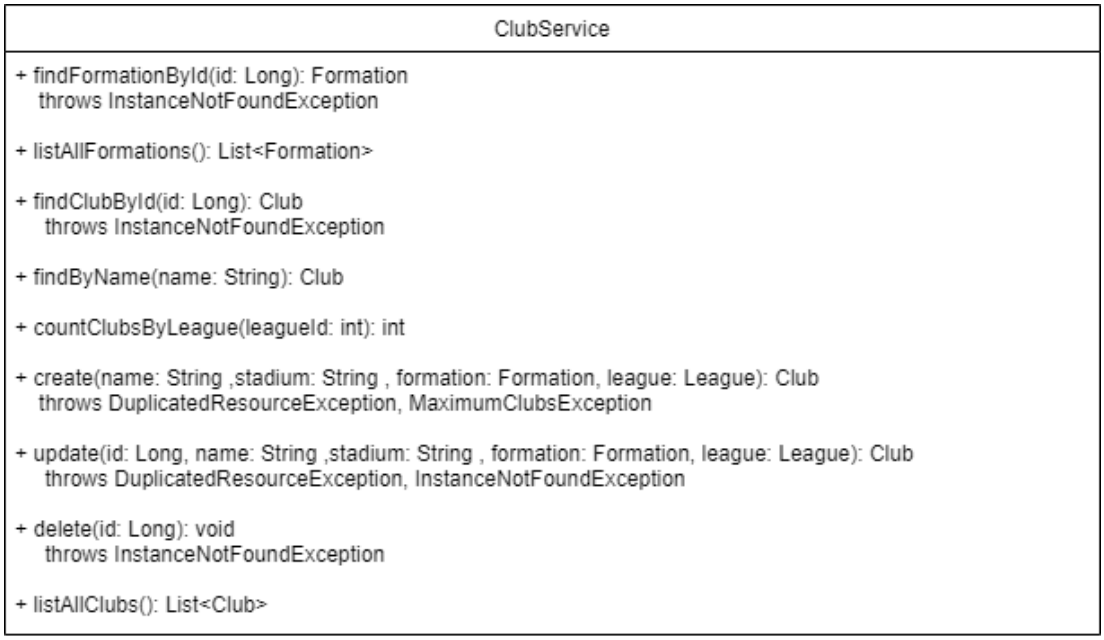

Figura 5.6: Diagrama de ClubService

- LeagueService: En esta clase se incluyen las funcionalidades relacionadas con las ligas como buscar una liga por nombre o id, añadir una nueva, modificarla o eliminarla.

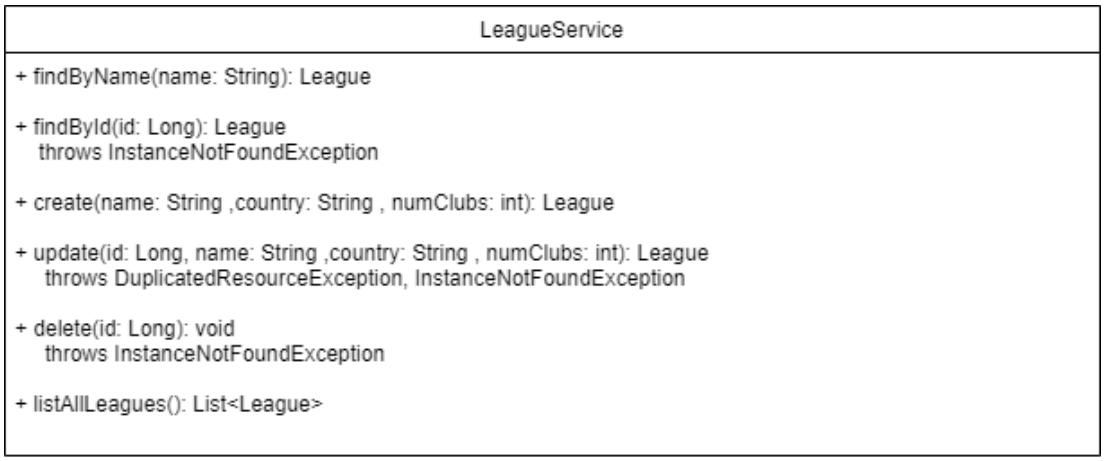

Figura 5.7: Diagrama de LeagueService

- MatchService: En esta clase se incluyen las funcionalidades relacionadas con los partidos, las jornadas, los eventos y las estadísticas como crear los partidos, jornadas, eventos y estadísticas, actualizar el partido, buscar los partidos disputados por un equipo, buscar por id, cambiar las posiciones de los jugadores antes de empezar un partido y guardar todas las estadísticas después de finalizar un partido.

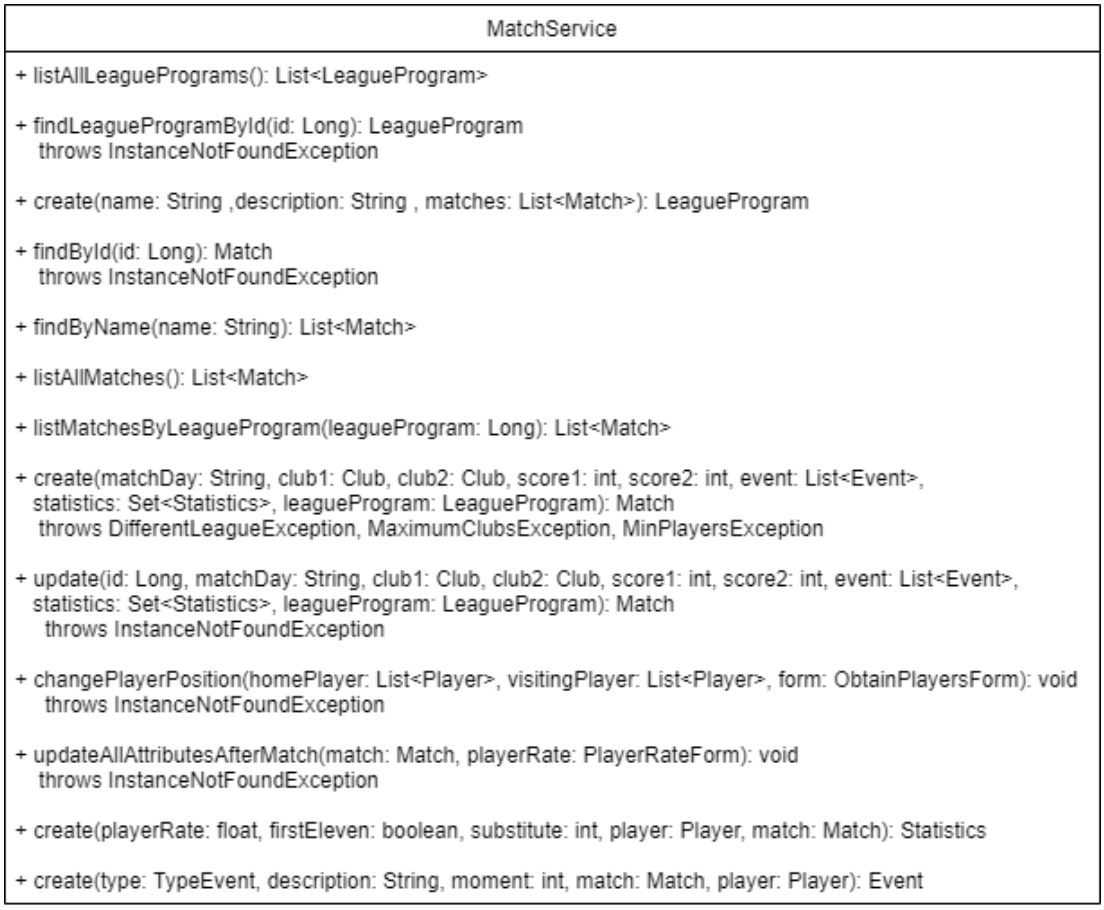

Figura 5.8: Diagrama de MatchService

Se va a explicar alguna de las funciones para que se entienda mejor, changePlayerPosition lo que hace es pasar la lista de jugadores locales y visitantes de los equipos que van a disputar el partido, y un formulario con las posiciones de cada uno de los jugadores, que son las posiciones que ha elegido el administrador y cuando termina la función estos cambios de posiciones son guardados en la base de datos. Y la función updateAllAtributesAfterMatch lo que hace es recolectar todos los eventos del partido (goles, asistencias, tarjetas...) puntuar a los jugadores por su relevancia en el partido, crear las estadísticas de cada jugador y actualizar los datos de los clubes para modificar la posición en la tabla de clasificación si fuese necesario.

- TournamentService: En esta clase se incluyen las funcionalidades relacionadas con la competición interna entre los usuarios como crear un torneo nuevo, actualizarlo, buscar por id, cambiar la posición de un jugador a la posición que el usuario quiera antes de un partido, que un usuario se pueda unir a un torneo y añadir los jugadores a un torneo.

Figura 5.9: Diagrama de TournamentService

Como pasaba en la anterior clase Service, aquí también se va a explicar alguna de las funciones para que se entienda mejor, registerTournament lo que hace es crear un nuevo torneo asociado a una liga en concreto en el sistema, joinUserToTournament lo que hace es que un usuario registrado en la aplicación se una un torneo ya creado seleccionado por el usuario y savePlayersToTournament es la función encargada de guardar en el usuario los jugadores con los que quiere participar ese usuario en el torneo.

Excepciones

A continuación se mostrarán las excepciones utilizadas para las funcionalidades de la aplicación.

- AuthenticationException: Se utiliza en relación a la autenticación de usuarios cuando hay algún problema con el login de usuario o el cambio de contraseña.
- DifferentLeagueException: Se utiliza cuando intentas crear un partido con dos clubes de diferentes ligas, ya que no tendría sentido este partido.
- DuplicatedResourceException: Se emplea cuando intentas crear una instancia con id ya existente.
- InstanceNotFoundException: Se emplea cuando intentas acceder a una instancia que no existe.

- MaximumClubsException: Se utiliza cuando intentas añadir un club a una liga que tiene ya el número máximo de clubes.
- MinPlayersException: Se emplea cuando intentas crear un partido entre dos clubes y alguno de ellos no tiene el número mínimo de jugadores para poder disputarlo.

5.3 Controlador

Para llevar a cabo el subsistema controlador se ha utilizado Spring, este subsistema es el encargado de gestionar las peticiones HTTP que llegan desde los clientes web, realizar las llamadas al modelo y finalmente devolver los resultados a la vista y los presente al usuario en formato HTML. Se ha implementado un controlador por cada Service, es decir 6 controladores, cada uno llamando a las funciones que hay en el modelo. Son las siguientes:

- UserController: este controlador se encarga de administrar las peticiones http relacionadas con los usuarios(inicio de sesión, registrarse, editar información del perfil...)
- PlayerController: esta clase es la encargada de gestionar las peticiones http relacionadas con los jugadores de fútbol como listar los jugadores, mostrar la información detallada de cada uno, registrar un nuevo jugador y eliminarlo entre otras.
- ClubController: esta clase se encarga de administrar las peticiones relacionadas con los clubes como añadir un nuevo club, modificarlo, mostrar la información del club, listar los clubes.
- LeagueController: este controlador se encarga de gestionar las peticiones relacionadas con las ligas como mostrar la información de una liga con una tabla de clasificación de clubes, una función para cuando finaliza la liga mostrando al ganador de la misma y listar las distintas ligas entre otras.
- MatchController: esta clase se encarga de las peticiones relacionadas con los partidos como obtener los jugadores que van a jugar el partido, el inicio del partido, registrar los eventos de un partido, establecer una puntuación a los jugadores que han jugado un partido y actualizar todas las estadísticas una vez finalice el partido.
- TournamentController: este controlador es el encargado de administrar las peticiones relacionadas con la competición interna entre usuarios como añadir un usuario a un torneo, seleccionar los jugadores de un usuario para participar en el torneo, cambiar las posiciones de los jugadores antes de un partido y mostrar los puntos que tiene cada usuario con respecto al resto.

Los controladores usarán DTOs para el intercambio de información, con esto se consigue independizar los subsistemas. A continuación se mostrarán los DTOs usados en este subsistema, todos nombrados con este formato ”xxxForm”:

- ChangePasswordForm
- ChangePlayerPositionForm
- ClubForm
- EventForm
- JoinTournamentForm
- LeagueForm
- LeagueProgramForm
- LoginForm
- MatchForm
- ObtainPlayersForm
- PlayerForm
- PlayerRateForm
- TeamTournamentForm
- TournamentForm
- UserCreateForm
- UserUpdateForm

A continuación se mostrarán los controladores con los principales mappings de las peticiones HTML, no se mostrarán todos ya que quedaría demasiado extenso.

5.3.1 UserController

/login	
HTTP	GET
Parámetros	
Descripción	Obtiene la pantalla de inicio de sesión en la aplicación

Tabla 5.1: GET /login

/login	
HTTP	POST
Parámetros	LoginForm
Descripción	Se envia el login y password una vez el usuario pulse en el botón ”Login”, si es correcto se inicia sesión y sino se muestra un mensaje de error

Tabla 5.2: POST /login

/loginGoogle	
HTTP	GET
Parámetros	
Descripción	Si en /login se pulsa en el botón ”Login with Google” te llevará a otra plantilla mostrando un botón para poder iniciar sesión con una cuenta registrada en Google

Tabla 5.3: GET /loginGoogle

/loginGoogle	
HTTP	POST
Parámetros	
Descripción	Si se pulsa el botón ”Signin with Google” se abrirá una ventana en la que podrás seleccionar la cuenta con la que quieras iniciar sesión, si todos los datos son válidos se iniciará sesión, sino se muestra un mensaje de error.

Tabla 5.4: POST /loginGoogle

/registration	
HTTP	GET
Parámetros	
Descripción	Se muestra la pantalla de registro para darse de al en la aplicación.

Tabla 5.5: GET /registration

/registration	
HTTP	POST
Parámetros	UserCreateForm
Descripción	Una vez el usuario rellena el formulario de registro y pulsa en el botón ”Register” se envían los datos, si todo está correcto el usuario se registra, sino se muestra un mensaje de error.

Tabla 5.6: POST /registration

/changePassword	
HTTP	GET
Parámetros	
Descripción	Se muestra la pantalla para que el usuario cambie su contraseña.

Tabla 5.7: GET /changePassword

/changePassword	
HTTP	POST
Parámetros	ChangePasswordForm
Descripción	Una vez el usuario rellene los campos solicitados y pulse en el botón ”Change Password” se enviarán los datos, si todos los datos son correctos se actualizará la contraseña, sino se mostrará un mensaje de error.

Tabla 5.8: POST /changePassword

5.3.2 PlayerController

/ListPlayers	
HTTP	GET
Parámetros	
Descripción	Muestra una lista con todos los jugadores registrados en la aplicación.

Tabla 5.9: GET /ListPlayers

/showPlayer/{playerId}	
HTTP	GET
Parámetros	playerId
Descripción	Muestra la información de un jugador concreto.

Tabla 5.10: GET /showPlayer/{playerId}

/createPlayer	
HTTP	GET
Parámetros	
Descripción	Se muestra la pantalla para registrar un nuevo jugador de fútbol.

Tabla 5.11: GET /createPlayer

/createPlayer	
HTTP	POST
Parámetros	PlayerForm
Descripción	Cuando el administrador rellena todos los datos para registrar un jugador y le da al botón ”Save” se envían los datos, si todo fue correcto el jugador se registra en la aplicación, sino se muestra un mensaje de error.

Tabla 5.12: POST /createPlayer

5.3.3 ClubController

/createClub	
HTTP	GET
Parámetros	
Descripción	Muestra la pantalla para registrar un nuevo club en la aplicación.

Tabla 5.13: GET /createClub

/createClub	
HTTP	POST
Parámetros	ClubForm
Descripción	Cuando el administrador rellena todos los datos para registrar un club y le da al botón "Save" se envían los datos, si todo fue correcto el club se registra en la aplicación, sino se muestra un mensaje de error.

Tabla 5.14: POST /createClub

/editClub/{clubId}	
HTTP	GET
Parámetros	clubId
Descripción	Muestra la pantalla con los datos registrados del club cuyo id se pasa por parámetros.

Tabla 5.15: GET /editClub/{clubId}

/editClub/{clubId}	
HTTP	POST
Parámetros	clubId, ClubForm
Descripción	Una vez el administrador modifica los datos y pulsa el botón "Save" se envían los datos, si todo es correcto se modifican los datos, sino se muestra un mensaje de error.

Tabla 5.16: POST /editClub/{clubId}

/showClub/{clubId}	
HTTP	GET
Parámetros	clubId
Descripción	Muestra la pantalla con la información detallada de un club concreto.

Tabla 5.17: GET /showClub/{clubId}

5.3.4 LeagueController

/league/{leagueId}/finishLeague	
HTTP	GET
Parámetros	leagueId
Descripción	Muestra la pantalla con la información detallada de la clasificación de una liga con el ganador de la misma una vez haya terminado.

Tabla 5.18: GET /league/{leagueId}/finishLeague

/createLeague	
HTTP	GET
Parámetros	
Descripción	Muestra la pantalla para registrar una nuevo liga en la aplicación.

Tabla 5.19: GET /createLeague

/createLeague	
HTTP	POST
Parámetros	LeagueForm
Descripción	Cuando el administrador rellena todos los datos para registrar una liga y le da al botón ”Save” se envían los datos, si todo fue correcto la liga se registra en la aplicación, sino se muestra un mensaje de error.

Tabla 5.20: POST /createLeague

/showLeague/{leagueId}	
HTTP	GET
Parámetros	leagueId
Descripción	Muestra la pantalla de la información detallada de una liga concreta.

Tabla 5.21: GET /showLeague/{leagueId}

5.3.5 MatchController

/createLeagueProgram	
HTTP	GET
Parámetros	
Descripción	Muestra la pantalla para registrar una jornada de liga.

Tabla 5.22: GET /createLeagueProgram

/createLeagueProgram	
HTTP	POST
Parámetros	LeagueProgramForm
Descripción	Cuando el administrador rellena todos los datos para registrar una jornada de liga y le da al botón ”Save” se envían los datos, si todo fue correcto la jornada se registra en la aplicación, sino se muestra un mensaje de error

Tabla 5.23: POST /createLeagueProgram

/showLeagueProgram/{leagueProgramId}/createMatch	
HTTP	GET
Parámetros	leagueProgramId
Descripción	Muestra la pantalla para registrar un partido en la aplicación.

Tabla 5.24: GET /showLeagueProgram/{leagueProgramId}/createMatch

/showLeagueProgram/{leagueProgramId}/createMatch	
HTTP	POST
Parámetros	leagueProgramId, MatchForm
Descripción	Cuando el administrador rellena todos los datos para registrar un partido y le da al botón "Save" se envían los datos, si todo fue correcto el partido se registra en la aplicación, sino se muestra un mensaje de error.

Tabla 5.25: POST /showLeagueProgram/{leagueProgramId}/createMatch

/showLeagueProgram/{leagueProgramId}/startMatch/{matchId}/obtainPlayers	
HTTP	GET
Parámetros	leagueProgramId, matchId
Descripción	Muestra la pantalla para la selección de jugadores y posición antes del partido.

Tabla 5.26: GET /showLeagueProgram/{leagueProgramId}/startMatch/{matchId}/obtainPlayers

/showLeagueProgram/{leagueProgramId}/startMatch/{matchId}/obtainPlayers	
HTTP	POST
Parámetros	leagueProgramId, matchId, ObtainPlayersForm
Descripción	Una seleccionamos los jugadores que van a jugar, comienza el partido.

Tabla 5.27: POST /showLeagueProgram/{leagueProgramId}/startMatch/{matchId}/obtainPlayers

/showLeagueProgram/{leagueProgramId}/startMatch/{matchId}	
HTTP	GET
Parámetros	leagueProgramId, matchId
Descripción	Muestra la pantalla principal del partido en donde se recogen los eventos que pasen a lo largo del partido(goles, asistencias, tarjetas...).

Tabla 5.28: GET /showLeagueProgram/{leagueProgramId}/startMatch/{matchId}

/showLeagueProgram/{leagueProgramId}/startMatch/{matchId}	
HTTP	POST
Parámetros	leagueProgramId, matchId, ObtainPlayersForm
Descripción	Cuando termina el partido se pulsa en el botón "Save" y se guardan todos los datos del partido en la aplicación.

Tabla 5.29: POST /showLeagueProgram/{leagueProgramId}/startMatch/{matchId}

5.3.6 TournamentController

/createTournament	
HTTP	GET
Parámetros	
Descripción	Muestra la pantalla para registrar un nuevo torneo en la aplicación.

Tabla 5.30: GET /createTournament

/createTournament	
HTTP	POST
Parámetros	TournamentForm
Descripción	Cuando el administrador rellena todos los datos para registrar un torneo y le da al botón ”Save” se envían los datos, si todo fue correcto el partido se registra en la aplicación, sino se muestra un mensaje de error.

Tabla 5.31: POST /createTournament

/joinTournament	
HTTP	GET
Parámetros	
Descripción	Muestra la pantalla para que el usuario pueda elegir a qué torneo unirse de los disponibles.

Tabla 5.32: GET /joinTournament

/joinTournament	
HTTP	POST
Parámetros	JoinTournamentForm
Descripción	Una vez el usuario elige el torneo al que quiere unirse y pulsa en el botón ”Save” se guarda en la base de datos y desde ese momento está inscrito en el torneo.

Tabla 5.33: POST /joinTournament

/selectPlayers	
HTTP	GET
Parámetros	
Descripción	Muestra la pantalla para que el usuario una vez se una al torneo pueda elegir los 11 jugadores con los que participará en el torneo.

Tabla 5.34: GET /selectPlayers

/selectPlayers	
HTTP	POST
Parámetros	TeamTournamentForm
Descripción	Una vez el usuario elige los jugadores con los que va a participar en el torneo y pulsa el botón ”Save” se guarda en la base de datos y ningún otro usuario podrá elegir los jugadores elegidos por este usuario.

Tabla 5.35: POST /selectPlayers

/changePlayerPosition	
HTTP	GET
Parámetros	
Descripción	Muestra la pantalla para que el usuario pueda elegir la posición de sus jugadores antes de un partido.

Tabla 5.36: GET /changePlayerPosition

/changePlayerPosition	
HTTP	POST
Parámetros	ChangePlayerPositionForm
Descripción	Una vez el usuario elige la posición de los jugadores antes del partido y pulsa el botón ”Save” se guarda en la base de datos.

Tabla 5.37: POST /changePlayerPosition

/showTournamentPoints	
HTTP	GET
Parámetros	
Descripción	Muestra la pantalla para que el usuario pueda ver los puntos de torneo que tiene con respecto a sus oponentes y la clasificación de los equipos que conforman la liga.

Tabla 5.38: GET /showTournamentPoints

5.4 Vista

Este subsistema es el encargado de ofrecer la interfaz web con la que el usuario interacciona y y accede a las funcionalidades de la aplicación.

En la figura 5.10 se muestra como es la secuencia de pasos que se produce desde que un cliente en la web hace una petición hasta que consigue la respuesta en la pantalla:

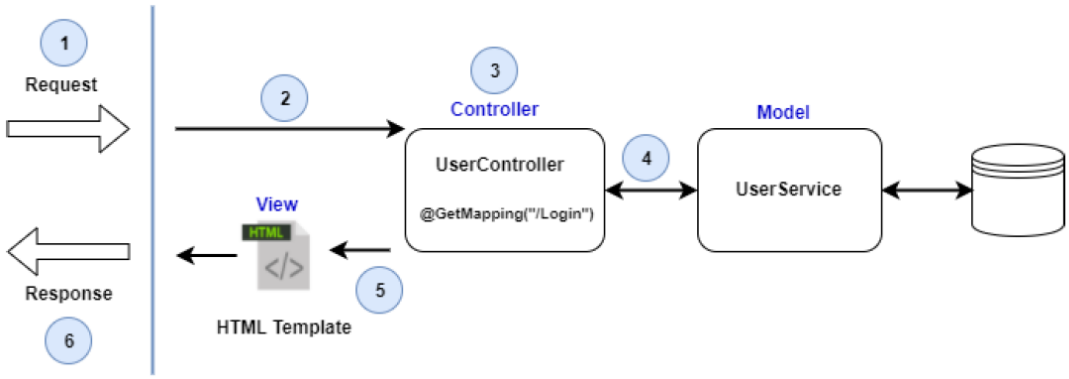

Figura 5.10: Flujo de interacciones

- 1. La petición se envía desde el cliente web.
- 2. Una vez que llega al servidor, en primer lugar se ejecutan los interceptores.
- 3. A continuación la petición llega al controlador.
- 4. El controlador accede a la lógica de negocio para obtener los datos necesarios que le permitan resolver la petición.
- 5. Se transfiere el control al motor de plantillas quien se encarga de la generación del contenido HTML.
- 6. Se envía la respuesta al cliente web.

5.4.1 Interfaz de usuario

La interfaz de usuario se ofrece a través de un conjunto de plantillas. Los usuarios de la aplicación dependiendo de donde hagan click crearán unos eventos determinados y serán enviados a los respectivos controladores donde el sistema recogerá los datos necesarios de la base de datos y devolverá esos datos a la página con la plantilla actualizada para que el usuario pueda ver el resultado final.

Las plantillas están implementadas en lenguaje HTML para mostrar los datos en la página web y los lenguajes CSS y JavaScript se encargarán de modificar los datos para una mejor experiencia para el usuario final de la aplicación. Para un desarrollo más fácil y rápido de los elementos más comunes de las páginas web se ha utilizado el framework de Bootstrap [8].

Todas las plantillas tienen una parte común de HTML la barra de navegación (Navbar) y el pié de página (footer). La barra de navegación tiene enlaces directos a recursos de la aplicación,

como por ejemplo, enlace al index de la aplicación, lista de ligas, jugadores, clubes y jornadas de ligas, y si el usuario está registrado un enlace a los torneos (la competición interna de la aplicación). En la parte derecha se muestran dos enlaces, uno para iniciar sesión y otro para registrarse en la aplicación, si el usuario ha iniciado sesión estos dos enlaces se cambiarán por otros dos, uno para el perfil del usuario y otro para cerrar sesión en la aplicación. El pié de página mostrará información sobre el proyecto y el correo del alumno.

5.4.2 Thymeleaf

Thymeleaf es un motor de plantillas, es decir, es una tecnología que nos va a permitir definir una plantilla con un gran nivel de detalle. Es importante tener un buen motor de plantillas ya que gracias a él se incluirán en las plantillas los datos del modelo. Thymeleaf también ofrece una buena integración con Spring Framework.

Como ya se dijo en el apartado anterior, todas las plantillas tienen una parte común, la barra de navegación y el pié de página. Todo esto se hace gracias a Thymeleaf, y se hace de una manera muy sencilla, en el apartado de implementación de la vista se mostrará cómo.

Capítulo 6

Implementación

6.1 Versiones del Software

A continuación se mostrará una lista con las versiones del software más importante utilizado para la implementación de la aplicación:

- **JDK**: 11.0.9.1_1
- **Apache Maven**: 3.6.3
- **Eclipse IDE for Enterprise Java and Web Developers**: 4.19.0
- **Google API Client**: 1.22.0
- **Spring Framework**: 2.0.5.RELEASE
- **Apache Derby**: 10.14.2.0
- **Thymeleaf**: 3.0
- **Git**: 2.23.0.windows.1

6.2 Estructura del proyecto

En la figura 6.1 se puede observar las estructuración del proyecto en eclipse y a continuación se explicará brevemente los paquetes más importantes:

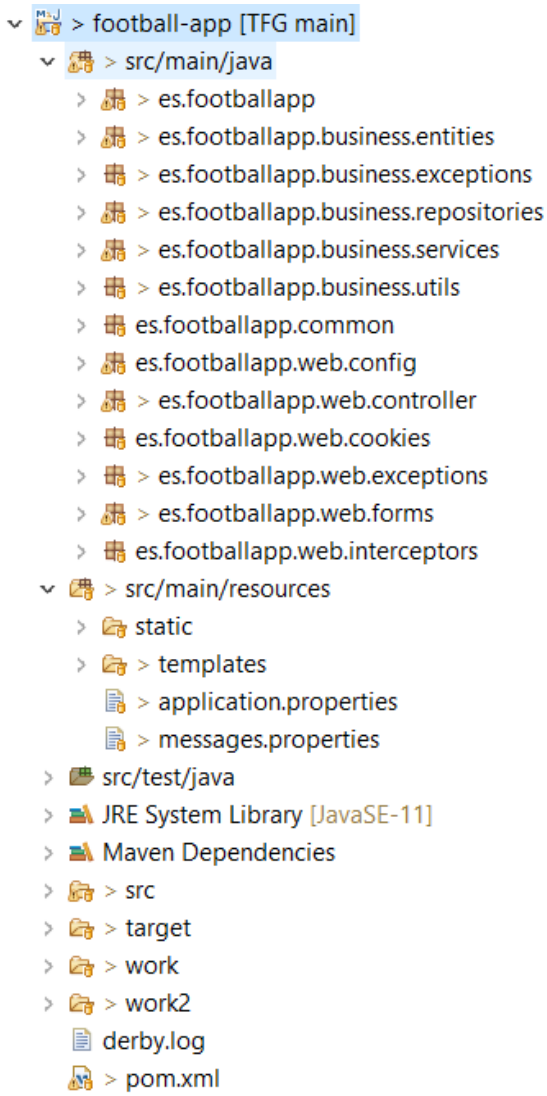

Figura 6.1: Estructura del proyecto

src/main/java

- En los paquetes "es.footballapp.buisiness.*" están las clases y código Java relacionado con la capa modelo, es decir, las entidades, repositorys, servicios y excepciones utilizadas en esta capa.

- En los paquetes "es.footballapp.web.*" están las clases relacionadas con los controladores, excepciones y formularios (DTOs) utilizados en esta capa

src/main/resources

- En el directorio ”static” están los ficheros .css y .js que ayudan a las plantillas a la hora de la representación de los datos y mejoras en la interfaz de usuario.
- En el directorio ”templates” están todas las plantillas HTML utilizadas en la capa vista.

6.3 Modelo

A continuación se hablará de los detallas de implementación de la lógica de negocios y servicio.

6.3.1 Lógica de negocios

Para implementar esta parte se ha utilizado JPA/Hibernate. Lo primero de lo que se va a hablar es de la implementación de las entidades y las relaciones.

El mapping de entidades a tablas se ha desarrollado por medio de las siguientes anotaciones:

- **@Entity**: indica que la clase es una entidad persistente, como mínimo una entidad tiene que tener una clave primaria y un constructor vacío sin argumentos.
- **@Table**: con la opción ”name” se utiliza para poner el nombre real de la tabla en la base de datos.
- **@Id**: indica la clave primaria de la entidad.
- **@GeneratedValue**: se utiliza para que Hibernate genere automáticamente el id.
- **@Column**: indicas el nombre que quieres que tenga ese atributo en la base de datos

Con respecto a las relaciones entre entidades tenemos las siguientes anotaciones:

- **@OneToMany**: Se utiliza en la entidad con cardinalidad uno en las relaciones 1:N.
- **@ManyToOne**: Se utiliza en la entidad con cardinalidad muchos en las relaciones N:1.
- **@OneToOne**: Se utiliza en las dos entidades en las relaciones 1:1.

En el diagrama de entidades 5.2 se da la relación N:M entre las entidades Match y Player, entonces en vez de utilizar la anotación @ManyToMany y que Hibernate cree una tabla intermedia, lo que se hizo fue hacerlo de forma manual y se creó una una entidad llamada

Statistics. Se decantó por esta opción ya que si se deja que lo haga Hibernate de forma automática, en la tabla intermedia no se podrían añadir nuevos atributos, solo las claves primarias que forman la tabla. Y como en este caso se necesitaban añadir distintos atributos se decidió por la opción de implementarlo de forma manual.

A continuación se mostrará una parte de una de las entidades como ejemplo en el que salen la mayoría de las anotaciones de las que se han hablado anteriormente.

```
@Entity
@Table(name = "Clubs")
public class Club implements Serializable{

    private static final long serialVersionUID = 1595960278705269842L;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long clubId;

    @Column(name = "name", nullable = false, unique = true)
    private String name;

    @Column(name = "stadium", nullable = false)
    private String stadium;

    @Column(name = "position", nullable = true)
    private int position;

    @ManyToOne
    @JoinColumn(name="formationId")
    private Formation formation;

    @Column(name= "num_players", nullable = false)
    private int numPlayers;

    @OneToMany(mappedBy = "club")
    private List<Player> players = new ArrayList<Player>();
```

Figura 6.2: ClubEntity

Los DAOs se han implementado como se había dicho en la parte del diseño, en esta aplicación se denominan ”xxxRepository”, la única anotación que se ha utilizado es @Repository, es una anotación de Spring framework que indica que la clase es un repositorio. Un repositorio es un mecanismo para encapsular el comportamiento de almacenamiento, recuperación y búsqueda que emula una colección de objetos. Es una especialización de la anotación @Component.
Hibernate utiliza un lenguaje de consulta denominado HQL que se parece a SQL. Sin embar-

go, comparado con SQL, HQL es completamente orientado a objetos y comprende nociones como herencia, polimorfismo y asociación. A continuación se mostrará un ejemplo de una clase repository con un ejemplo de una función que busca clubes por nombre y utiliza una consulta HQL sencilla.

```
private static final String FIND_CLUB_BY_NAME ="SELECT l FROM Club l WHERE l.name = ''{0}''";


public Club findClubByName(String name) {
    try {
        Query query = entityManager.createQuery(MessageFormat.format(FIND_CLUB_BY_NAME, name));
        return (Club) query.getSingleResult();
    } catch (NoResultException e) {
        return null;
    }
}
```

Figura 6.3: Ejemplo HQL en ClubRepository

6.3.2 Servicio

En la implementación del servicio se van a hacer uso de las funciones de los DAOs (Repository) mencionadas anteriormente, estos servicios son los encargados de la lógica de negocio de la aplicación.
Las anotaciones que se han utilizado en el servicio son las siguientes:

- **@Service**: indica que esta clase es un servicio.
- **@Autowired**: hace que Spring pueda inyectar unas dependencias con otras.
- **@Transactional**: esto indica que todas las funciones llamadas en el método que tenga esta anotación se van a producir en una misma transacción, es decir, si todas las funciones se realizan de forma correcta sin ninguna excepción al finalizar ser produce un commit y se cierra la transacción pero si en alguna de las llamadas se produce algún fallo, entonces se producirá un rollback y si desharán todos los cambios en la transacción. Además si la función en la que utilizas la anotación @Transactional es de solo lectura puedes añadir la opción ”readOnly = true” para que Spring pueda hacer optimizaciones en la transacción.

A continuación en la figura 6.4 se mostrará un ejemplo de la clase UserService en el que se utilizan las anotaciones anteriormente mencionadas.

```java
@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    ExceptionGenerationUtils exceptionGenerationUtils;

    @Transactional(readOnly = true)
    public User findByEmail(String email) {
        return userRepository.findByEmail(email);
    }

    @Transactional(readOnly = true)
    public User login(String email, String clearPassword) throws AuthenticationException {
        if (!userRepository.existsUser(email)) {
            throw exceptionGenerationUtils.toAuthenticationException("auth.invalid.user", email);
        }
        User user = userRepository.findByEmail(email);
        boolean isPasswordCorrect = BCrypt.checkpw(clearPassword, user.getPassword());
        if (!isPasswordCorrect) {
            throw exceptionGenerationUtils.toAuthenticationException("auth.invalid.password", email);
            }
        return user;
    }
```

Figura 6.4: Ejemplo UserService

6.4 Controlador

En este apartado se va a hablar de como se han implementado los distintos controladores que recibirán las peticiones HTTP realizadas en la vista por los usuarios de la aplicación. Se han usado las siguientes anotaciones de Spring Framework:

- **@Controller**: indica que la clase es un bean controlador y nos ahorra el trabajo de definir en XML el bean correspondiente.
- **@Autowired**: como ya se dijo en el apartado anterior, esta anotación se utiliza para resolver las dependencias, en este caso para poder usar las funciones del servicio en los diferentes controladores.
- **@GetMapping**: permite asociar la URL que pasas como parámetro a la función y además añade la información de que el método HTTP utilizado es un GET, esta anotación es quizás más conocida de esta forma: @RequestMapping(method = RequestMethod.GET).
- **@PostMapping**: permite asociar la URL que pasas como parámetro a la función y además añade la información de que el método HTTP utilizado es un POST, esta anotación es quizás más conocida de esta forma: @RequestMapping(method = RequestMethod.POST).

A continuación en la figura 6.5 se mostrará un trozo de código implementando la clase UserController con su correspondiente explicación.

```java
@Controller
public class UserController {

    private static final Logger logger = LoggerFactory.getLogger(UserController.class);

    @Autowired
    private MessageSource messageSource;

    @Autowired
    private UserService userService;

    @Autowired
    ErrorHandlingUtils errorHandlingUtils;

    @GetMapping("/login")
    public String doGetLoginPage(Model model) {
        model.addAttribute("loginForm", new LoginForm());
        return "Login";
    }

    @PostMapping("/login")
    public String doLogin(@Valid @ModelAttribute LoginForm loginForm,
                          BindingResult result,
                          @RequestParam(value = "next", required = false) String next,
                          HttpSession session,
                          HttpServletResponse response,
                          Locale locale,
                          Model model) {
        if (result.hasErrors()) {
            errorHandlingUtils.handleInvalidFormError(result,
                    "registration.invalid.parameters", model, locale);
            return "Profile";
        }
```

Figura 6.5: Ejemplo UserController

Como se puede observar en la figura anterior, existen dos funciones distintas para la misma URL pero una con la anotación @GetMapping y otra con @PostMapping. Esto es porque la función con el método GET sirve para enviar los datos necesarios a la vista para que el usuario pueda ver cierta información, en este caso un formulario de login en el que el usuario verá un formulario para poder loggearse en la aplicación. La función con el método POST sirve para cuando el usuario cubra el formulario de login y pulse en el botón de submit esos datos pasan a esta función y se procesan para comprobar si los datos introducidos son correctos o contienen algún error. Si sale error se mostrará en la pantalla un mensaje de error y sino el usuario iniciará sesión en la aplicación. Todo esto pasa con todas las URL en las que se necesita la interacción con el usuario, si por ejemplo es una vista en la que se muestra la lista de equipos de una liga, entonces solo se implementaría una función con el método GET.

6.5 Vista

En la implementación de la vista se ha utilizado el lenguaje HTML y el motor de plantillas Thymeleaf. Para dar una mejor visualización a los datos de las plantillas se ha ayudado de ficheros CSS y JavaScript para mejoras de dinamismo.

Como ya se ha dicho en el apartado de diseño, todas las plantillas tienen una parte común: la barra de navegación y el pié de página. Esto es posible gracias al motor de plantillas Thymeleaf. Pero antes de explicar cómo utilizar los atributos de Thymeleaf en las plantillas de la aplicación, siempre que se quiera utilizar Thymeleaf se tiene que añadir el prefijo ”th:” y a parte añadir el namespace en las plantillas que se quiera utilizar este motor de plantillas ”<html xmlns:th=”http://www.thymeleaf.org”>”. En la siguiente figura 6.6 se mostrará un ejemplo de la parte común que tiene cada una de las plantillas creadas en la aplicación.

```
<!DOCTYPE HTML>
<html>

   <header th:replace="layout/Includes :: #include-fragment"></header>

   <body>

      <nav th:replace="layout/NavBar :: #navbar-fragment"></nav>
      <div th:replace="layout/Messages :: #messages-fragment"></div>

      <div class="container">

            ...

      </div>

      <div th:replace="layout/Footer :: #footer-fragment"></div>
   </body>
</html>
```

Figura 6.6: Ejemplo Plantilla

En la imagen se ha suprimido toda la información relacionada con la representación de datos para poder explicar todo con mayor claridad. La expresión ”th:replace” lo que hace es reemplazar el código que hay en el fichero ”layaout/Includes” por la etiqueta <header> que tenemos en la plantilla y lo mismo pasa con los otros 4 ”replace”. Cada uno es una plantilla diferente:

- **Includes**: en esta plantilla están todos los includes necesarios para poder representar los datos y utilizar distintas librerías, por ejemplo: JavaScript, CSS, Bootstrap, Thyme-

leaf...

- **NavBar**: aquí se encuentra el código relacionado con la barra de navegación y como se dijo anteriormente está en todas las plantillas.
- **Messages** : en esta plantilla se encuentra el código para poder mostrar todos los mensajes del tipo success, warning y error de los eventos de la aplicación.
- **Footer**: esto es el pié de página de la aplicación y como pasa con la barra de navegación está en todas las plantillas.

Pues como se puede observar, gracias a Thymeleaf podemos incluir todo este código sin necesidad de cargar las plantillas con las mismas líneas de código en todas ellas y así simplificar mucho las plantillas.

Capítulo 7

Pruebas

Para asegurarse del correcto funcionamiento de la aplicación se han implementado distintos tipos de pruebas, de aceptación y funcionales. También se tuvo en mente implementar las pruebas unitarias pero debido a la falta de tiempo no se ha podido llevar a cabo. A continuación se explicarán las pruebas implementadas.

7.1 Pruebas funcionales

Para probar el correcto funcionamiento de la aplicación se han realizado pruebas funcionales, estas pruebas se realizan a través de la interfaz web simulando un usuario final. Se han probado una a una todas las funcionalidades de la aplicación y sus respectivas excepciones, se realizan de forma manual y con la ayuda de Eclipse y su modo ”debug” se ha podido observar que todos los pasos intermedios y todos los datos son correctos a lo largo de la distintas capas, desde que un usuario crea un evento en la vista, pasa por el controlador, el controlador llama a la función asociada a ese caso de uso en el servicio, este obtiene los datos necesarios de la base de datos y devuelve el control a la vista con la plantilla actualizada para representar los datos necesarios. En todos los casos se ha intentado lo siguiente:

- Probar los casos de uso en distinto orden. Un ejemplo sería: registrar a un usuario y que se una a un torneo, cierre sesión y que al volver a iniciarla siga unido a ese torneo y que no le deje unirse a otro torneo. Otro que si un usuario aún no está registrado en la aplicación no le permita unirse a un torneo, ya que es una competición interna entre los usuarios de la aplicación. Si por ejemplo un usuario ya ha iniciado sesión se le permita cerrar la sesión y no le deje iniciar otra vez la sesión (no tendría sentido).

- Probar un mismo caso de uso varias veces. Si por ejemplo se quiere buscar un jugador por nombre, que pasando el mismo nombre aparezca el mismo resultado. O si una secuencia de acciones produce una excepción, que haciendo las mismas acciones se pro-

duzca la misma excepción. Si por ejemplo estás buscando una lista de clubes y se añade un club nuevo, la próxima vez que realices ese caso de uso se muestre ese nuevo club.

- Probar en todos los formularios de creación de entidades(clubes, jugadores, usuarios...) que la validación funciona correctamente, es decir, si en un campo tienes que meter números que no se permita meter letras y de un error acorde al fallo, si en un campo no puede haber espacios entre los caracteres que se cumpla, o si por ejemplo cuando un usuario tiene que añadir su teléfono que cumpla el patrón.

7.2 Pruebas de aceptación

Las pruebas de aceptación son uno de los últimos pasos antes de entregar la aplicación al cliente, se realizan para determinar si los requerimientos del cliente han sido cumplidos respecto a lo que pidió al inicio del proyecto. Estas pruebas son fundamentales para asegurar el éxito de la implementación final de un proyecto de ingeniería de software. Han sido realizadas por el alumno y como se dijo anteriormente en ellas se valida el correcto funcionamiento y los requisitos iniciales de la aplicación.

Capítulo 8

Planificación y evaluación de costes

8.1 Planificación

En este apartado se va a mostrar la planificación de los sprints, es decir, la duración en fechas y horas que se ha necesitado para implementar cada uno de ellos. A partir de esto se ha calculado el coste del proyecto.
A continuación se mostrará una tabla 8.1 con la duración de cada sprint desde el día que empezó hasta el que se finalizó y las horas que se ha dedicado a cada uno de ellos con una breve explicación de cada uno.

Sprint	Inicio	Fin	Días	Horas
0	18/02/21	24/03/21	34	170
1	25/03/21	11/04/21	17	85
2	12/04/21	25/04/21	13	65
3	26/04/21	10/05/21	14	70
4	11/05/21	29/05/21	18	90
5	30/05/21	20/06/21	21	105

Tabla 8.1: Fechas y duración de cada sprint

8.1.1 Sprint 0

Como se puede observar en los datos de la tabla anterior, este es el sprint con mayor duración, y esto es debido a que a parte del tiempo de implementación de las historias de usuario, el alumno tuvo que emplear bastante tiempo en el aprendizaje de las nuevas tecnologías que se han usado en este proyecto. Con respecto a las historias de usuario, en este sprint se han implementado las funcionalidades relacionadas con el usuario.

8.1.2 Sprint 1

En este sprint la duración se ha reducido considerablemente debido a que el alumno ya estaba familiarizado con el entorno y costaba menos a la hora de la implementación del sprint. Con respecto a las historias de usuario, en este sprint se han implementado las funcionalidades relacionadas con las ligas y los clubes.

8.1.3 Sprint 2

En este sprint la duración se ha reducido ligeramente respecto al anterior sprint, el motivo por el que pasa esto es el mismo que en el anterior sprint, el alumno tarda menos en implementar las funcionalidades debido a la familiaridad con el entorno de trabajo. Con respecto a las historias de usuario, se han implementado las funcionalidades relacionadas con los jugadores y las jornadas de liga.

8.1.4 Sprint 3

El esfuerzo en este sprint es prácticamente igual que el anterior, debido a las mismas razones que se comentaron antes. Con respecto a las historias de usuario, se han implementado las funcionalidades relacionadas con los partidos.

8.1.5 Sprint 4

En este sprint ha sido ligeramente superior a los anteriores, esto ha ocurrido porque el alumno ha tenido algunos problemas a la hora de implementar las funciones relacionadas con el torneo. Este sprint no debería haber sido tan largo.

8.1.6 Sprint 5

Este sprint ha sido el segundo de más duración, esto es porque a parte de implementar las 3 historias de usuario previstas y que una de ellas se ha complicado (inicio de sesión con Google), también se ha incluido las horas que se ha tardado en redactar la memoria del proyecto.

8.2 Coste

El hardware que se ha utilizado para la realización del proyecto, un ordenador portátil, es propiedad del alumno. Por tanto no se añadirá el coste del portátil en el coste total del proyecto. El coste total del proyecto se corresponde con los recursos humanos. Se estimó que el coste del alumno es de 25 €/hora y un coste de 45 €/hora para el jefe del proyecto (profesor). Cada semana desde que se empezó el proyecto, el alumno y el profesor se han reunido una

hora a la semana a través de la plataforma Microsoft Teams, la duración del proyecto ha sido de unas 20 semanas por tanto el profesor a hecho un esfuerzo de 20 horas y la suma de todas las horas de los sprints da que el alumno ha tenido un esfuerzo de 585 horas. En la siguiente tabla 8.2 se muestra el coste total del proyecto:

Recurso	**Horas**	**Coste (€/Hora)**	**Coste (€)**
Alumno	585	25	14.625
Jefe de proyecto	20	45	900
Coste total del proyecto			15.525

Tabla 8.2: Coste total del proyecto

El coste estimado en la tabla anterior sería calculándolo sin aplicar el IVA, que es de unos 15.525€, aplicando el IVA del 21€ subiría a un coste estimado de 18.785,25 €.

Capítulo 9

Conclusiones y líneas futuras

Como ya se dijo en el apartado de introducción, el objetivo de este proyecto es analizar, diseñar e implementar un aplicación web Java para gestionar las ligas de fútbol y una competición de equipos virtuales entre los usuario de la aplicación. A continuación se mostrarán las conclusiones y lineas futuras una vez terminado el proyecto.

9.1 Conclusiones

Tras finalizar el proyecto, se puede concluir que se han alcanzado los objetivos que se fijaron al inicio del proyecto, implementando una página web para los amantes del fútbol en el que puedes consultar los datos de cualquier futbolista de las principales ligas europeas, sus respectivos clubes, la clasificación de los mismos, los resultados en cada jornada de liga y también un torneo interno en el que puede participar cualquier usuario registrado en la web.

Personalmente querría comentar algunos aspectos enumerados a continuación:

- Lo primero que quiero decir es que realizar este TFG me ha servido conocer nuevos conceptos relacionados con las páginas web, como puede ser las plantillas HTML ayudado por el motor de plantillas Thymeleaf, el framework Spring, Hibernate e incluso la modificación de los datos en las plantillas gracias a CSS. También a la hora de realizar la memoria, el editor de texto Latex, que no lo conocía y me he dado cuenta de las ventajas que puede tener a la hora de realizar una memoria bien estructurada.

- Al tener que realizar un proyecto de forma individual, al contrario que en la carrera, me he dado cuenta que tienes que tener muy claro todos los aspectos del proyecto e investigar mucho más para poder llegar al objetivo final, por el contrario si son varios en el proyecto se reparten las tareas y se suele terminar antes pero no acabas con los mismos conocimientos.

- En la fase inicial del proyecto, cuando tienes que elegir el tema sobre el que vas a hacer el proyecto me ha resultado bastante complicado saber exactamente lo que quería, ya que normalmente a lo largo de la carrera siempre tenías un enunciado en el que apoyarte y sabes exactamente lo que tienes que hacer. Pero en este caso al ser uno mismo el que lo tiene que redactar me he dado cuenta porqué nos decían en la carrera la frase ”el cliente muchas veces no sabe realmente lo que quiere”.
- Por último, comentar que terminar este proyecto ha sido una satisfacción personal muy grande ya que hace unos años en la carrera pensar en terminar el TFG parecía algo imposible.

9.2 Líneas futuras

Como pasa siempre en todos los proyectos, la primera versión está abierta a nuevas funcionalidades y mejoras en la aplicación y esta no iba a ser una excepción. A continuación se enumerará un lista de aspectos de mejora:

- Mejoras visuales en la interfaz de usuario, es decir, mejor utilización del código HTML y con la ayuda de la librería de Bootstrap y también mejoras dinámicas utilizando JavaScript para una mejor experiencia de usuario.
- Añadir imágenes en la aplicación para jugadores de fútbol, usuarios, escudos de los equipos de fútbol... Y así asociar mejor los recursos de la aplicación con una imagen real del mismo.
- Utilizar redes sociales (Facebook, Twitter...) a la hora del inicio de sesión de un usuario, a parte de la opción que se ha implementado con Google.
- Implementar la internacionalización de la aplicación, es decir, añadir nuevos idiomas para los usuarios finales y así que más usuarios puedan utilizar la aplicación.
- Implementar las pruebas unitarias, ya que como se comentó en el apartado de pruebas, por falta de tiempo no se pudieron implementar.

Lista de acrónimos

API Application Programming Interface, interfaz de programación de aplicaciones. 4, 15

CRUD Create Read Update Delete, operaciones básicas de acceso a datos (crear, leer, actualizar y eliminar).. 12, 13

CSS Cascading Style Sheets, hojas de estilo en cascada. 4, 34, 44, 53

DAO Data Access Object, objeto de acceso a datos. 40, 41

DTO Data Transfer Object, objeto de transferencia de datos. 24, 38

HQL Hibernate Query Languaje, lenguaje de consultas de Hibernate. 40, 41

HTML HyperText Markup Language, lenguaje de marcas de hipertexto. 1, 4, 15, 24, 34, 39, 44, 53, 54

HTTP Hypertext Transfer Protocol, Protocolo de transferencia de hipertexto. 15, 42

IDE Integrated Development Environment, entorno de desarrollo integrado. 5

JDT Java Development Toolkit, desarrollo de programas Java. 5

JPA Java Persistence API, API de persistencia de Java. 15

JSP JavaServer Pages. 5

MVC Model View Controller, modelo vista controlador. 1, 5, 12, 15

POO Object-Oriented Programming, programación orientada a objetos. 4

SQL Structured Query Language, lenguaje de consulta estructurada. 40, 41

URL Uniform Resource Locator, localizador de recursos uniforme. 42, 43

XML eXtensible Markup Language, lenguaje de marcado extensible. 4, 5

Bibliografía

[1] "Documentacion de java." [En línea]. Disponible en: https://www.java.com/

[2] "Documentacion de javascript." [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/JavaScript

[3] "Documentacion de hibernate." [En línea]. Disponible en: https://hibernate.org/

[4] "Documentacion de spring." [En línea]. Disponible en: https://spring.io/

[5] "Documentacion de thymeleaf." [En línea]. Disponible en: https://www.thymeleaf.org/

[6] "Documentacion de eclipse." [En línea]. Disponible en: https://www.eclipse.org/

[7] "Documentacion de apache maven." [En línea]. Disponible en: https://maven.apache.org/

[8] "Documentacion de bootstrap." [En línea]. Disponible en: https://getbootstrap.com/