Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE GRADO

TÍTULO DEL TFG: Generador de competiciones en una herramienta de gamificación

TITULACIÓN: Grado en Ingeniería de aeronavegación

AUTOR: Mª del Mar Olmo Gutiérrez

DIRECTOR: Miguel Valero García

FECHA: 6 de julio del 2018

Título: Generador de competiciones en una herramienta de gamificación

Autor: Mª del Mar Olmo Gutiérrez

Director: Miguel Valero García

Fecha: 6 de julio del 2018

Resumen

El proyecto descrito en este documento forma parte de una línea de proyectos destinada a implementar una aplicación web y móvil, denominada Classpip, cuya finalidad es gamificar la docencia. Esta línea de proyectos se desarrolla sobre una plataforma de software de código abierto ya existente, la cual asienta las bases desde las que se parte y las tecnologías que se usarán, entre las que destacan la tecnología HTML5 o el lenguaje JavaScript. Todo el desarrollo de la aplicación se ha realizado con tecnologías de código abierto.

El objetivo de este proyecto es el desarrollo de una herramienta dedicada a la creación de competiciones entre alumnos y su posterior gestión por parte de los profesores a través del panel de administración web. Se incluye una herramienta adicional que permite dividir a los estudiantes de una determinada clase en tantos equipos como el profesor desee, para así, incorporar la modalidad de equipos a las competiciones. Por último, esta nueva sección de competiciones que se va a incorporar, también se implementa en la aplicación móvil, aunque desde ésta sólo se podrá acceder a modo de consulta.

Para gestionar las peticiones de ambos frontales (web y móvil), existe una arquitectura orientada a servicios con la lógica necesaria para actúar como único back-end. Contiene una base de datos MySQL, para almacenar toda la información recibida, y una API (interfaz de programación de aplicaciones) que se encarga de la comunicación entre los diferentes componentes del software.

El panel de administración se desarrolla mediante Angular, pudiendo ser utilizado desde cualquier sistema operativo. La aplicación móvil se crea mediante Ionic, basado en Angular, para el desarrollo de aplicaciones híbridas, que junto con Apache Cordova permite la conversión de la plataforma nativa a Android e iOS utilizando tecnologías web estándar.

Para conseguir el desarrollo colaborativo, todos los componentes de la aplicación se alojan en repositorios de la plataforma GitHub, desde donde además se comprueba el correcto desarrollo del código y su posible integración en el código fuente inicial.

Principalmente, este documento se centra en analizar y describir con detalle todas las funcionalidades que se han incluido en el panel de administración y en la aplicación móvil. Además, explica lo esencial para comprender el código implementado facilitando el trabajo a futuros desarrolladores que continúen con esta herramienta, e incluye una validación de la aplicación, que examina sus funcionalidades a través de un videotutorial y realiza una evaluación de usuario que aporta nuevas ideas y mejoras para continuar creciendo.

Title: Competition generator in a gamification tool

Author: Mª del Mar Olmo Gutiérrez

Director: Miguel Valero García

Date: July 6 th 2018

Overview

The project described in this work is a part of a collaborative project with the aim of implementing a web and mobile app, named Classpip, whose objective is to gamify teaching. This project is developed on a pre-existing open source software platform that sets the foundations and the technologies that will be used in this project, among which the HTML5 technology or the JavaScript language stand out. The whole process of development of the app has been performed with open source technologies.

The aim of this project is the development of a tool dedicated to the creation of competitions between students and their posterior management by their teachers through a web admin portal. An additional tool that organizes students into teams is included, allowing to create team competitions. Lastly, this new competition section is also implemented in the mobile app, although only in query mode.

To manage requests from both fronts (web and mobile), there is service-oriented architecture with the necessary logic to act as a lone back-end. It contains a MySQL database to store all of the received information and an API (Application Programming Interface) that is in charge of communication between the different components of the software.

The admin panel is developed using Angular, and can be used from any operating system. The mobile app is created with Ionic, which is based on Angular, for the development of hybrid apps, which together with Apache Cordova allows the conversion of the native platform to Android and iOS using standard web technologies.

To achieve collaborative development, all the components of the app are deposited in the GitHub platform repository, where the correct development of the code and its possible integration in the original source code are checked.

Mainly, this document focuses on analyzing and describing in detail all the functionalities that have been included in the admin panel and in the mobile app. In addition, it explains the essentials to understand the implemented code, facilitating the job of future developers that continue working on this tool. It also includes an app validation that examines its functionalities through a videotutorial and carries out a user assessment that brings new ideas and improvements to continue growing.

A Miguel Valero,
por haberme guiado y ayudado en todo momento
durante la realización de este proyecto.

A mi familia,
por todo el ánimo y el cariño incondicional que me han dado.

A Miguel Delgado,
por haber estado a mi lado durante todo este tiempo
ayudándome, aconsejándome y dándome su apoyo y cariño.

A Sandra Bordera,
por haber sido mi compañera y mi máximo apoyo
desde el primer día de universidad hasta el último.

Y por último, a todas las personas
implicadas en esta aplicación, por estar siempre dispuestos a ayudarte en todo.

ÍNDICE

ÍNDICE DE FIGURAS Y TABLAS

INTRODUCCIÓN

A día de hoy, existen diferentes metodologías de enseñanza cuyo único objetivo es mejorar el rendimiento académico de los alumnos. La gamificación es una de las técnicas con la que mejores resultados se obtienen; se basa en utilizar la mecánica de los juegos en entornos no lúdicos, como son los centros docentes, con el propósito de motivar al alumno a desear aprender y superarse. Una buena forma de lograr establecer esta metodología en el aprendizaje se consigue con la ayuda de aplicaciones diseñadas para ello.

Con lo cual, este proyecto persigue contribuir al desarrollo de una aplicación destinada a la gamificación de entornos docentes. Para lograrlo, existe ya una plataforma de software de código abierto que asienta las bases y las tecnologías implicadas en esta aplicación y sobre la cual, se van implementando diversos proyectos que aportan las diferentes funcionalidades que se desea que tenga (una herramienta para la asignación de puntos, cartas e insignias como recompensa, un módulo que permita crear preguntas para los estudiantes, etc.).

Por lo tanto, el **objetivo** principal de este proyecto, es el desarrollo de una nueva **sección dedicada a las competiciones**. Se centra en la creación de dos tipos diferentes de competición, liga y torneo de tenis, donde los ganadores no se conocen en un mismo día sino al cabo de varias jornadas, consiguiendo así, mantener el espíritu competitivo de los alumnos siempre activo y querer mejorar para alcanzar la victoria. En este espacio, los profesores podrán crear y gestionar competiciones desde un panel de administración web. Además, también se incluirán todas las funcionalidades de consulta en la aplicación móvil.

Al crear las competiciones, además de establecer su tipo y otros parámetros, será posible determinar su modalidad: individual o por equipos. Aunque para conseguir esta última se debe crear una herramienta adicional que divida a los estudiantes en equipos, por lo que también se desarrollará en el panel de administración este generador de equipos.

Para la gestión de las competiciones, el profesor podrá determinar las fechas de las jornadas existentes, añadir un texto informativo acerca de la competición e introducir los resultados de los enfrentamientos tanto de forma manual como aleatoria, añadiendo con esta última un elemento de azar que podría hacer la competición más interesante.

Profesores y alumnos, podrán informarse en todo momento del estado en el que se encuentra la competición. Existirán varias secciones para consultar: un calendario donde ver los emparejamientos, resultados y fechas de cada jornada, una sección de equipos (solo modalidad de equipos) para recordar qué alumnos pertenecen a cada equipo, un panel desplegable con la información dada por el profesor, una clasificación (sólo liga) donde revisar la puntuación, duelos ganados, etc. de cada estudiante, y por último, un seguimiento del torneo (sólo torneo de tenis) para comprobar en qué estado se encuentran los estudiantes en ese momento (si han sido eliminados, si permanecen en el torneo principal o si pertenecen al secundario), ya que se trata de un torneo de doble eliminación.

Para lograr estos objetivos, el panel de administración se desarrolla mediante **Angular 4**, permitiendo su uso en cualquier sistema operativo y escrito en lenguaje TypeScript. Para la aplicación móvil, en cambio, se utiliza **Ionic 2**, que junto con Apache Cordova, es posible el desarrollo de aplicaciones híbridas y la conversión de la plataforma nativa a Android e iOS evitando así, tener que crear código exclusivo para cada uno, simplemente utilizando tecnologías web estándar. Además, Ionic está basado en Angular, por lo que es posible la reutilización de código y paquetes entre ambos.

Tanto el panel de administración como el móvil, se comunican con un mismo back-end, que se trata de una arquitectura orientada a servicios encargada de tramitar todas las peticiones que realiza el usuario. Contiene una interfaz de programación de aplicaciones (API) que permite la comunicación entre los diferentes componentes del software, por lo que los datos pueden ser proporcionados a las diferentes plataformas del front-end a la vez que éstas pueden enviar nuevos datos o modificaciones al servidor para guardarlas en una base de datos MySQL. Para realizar estas operaciones con la base de datos es necesaria la creación de nuevos modelos y relaciones para así guardar la información.

Para comprobar el correcto funcionamiento de la lógica que se va desarrollando en cada uno de estos componentes, deben de ir ejecutándose pruebas que validen su funcionalidad con la finalidad de su poder efectuar su integración en el código base original.

Por tanto, este documento se ha dividido en capítulos que siguen un orden lógico para facilitar la comprensión de todo el contenido. Comienza explicando y asentando las bases desde las que se parte y todas las tecnologías implicadas. Continúa analizando con detalle los objetivos a alcancar desde una perspectiva funcional y visual para, seguidamente, explicar cómo se han conseguido. Finalmente, se realiza una validación de todo lo añadido en la aplicación.

CAPÍTULO 1. GAMIFICACIÓN

En el primer capítulo se hará primeramente una introducción al concepto de gamificación, así como una breve explicación del estado del arte actual. A continuación, se definirán los objetivos, principales aplicaciones y beneficios de la gamificación en el contexto actual. Finalmente, se concluirá el capítulo con un enfoque de la gamificación en el ámbito educativo.

1.1. Definición y objetivo

La **gamificación** es una técnica de aprendizaje que utiliza la mecánica de los juegos con la finalidad de conseguir mejores resultados en el ámbito educativo-profesional, ya sea para adquirir nuevos conocimientos o mejorar aptitudes ya adquiridas.

Según dos de los más importantes autores que estudian la gamificación, Gabe Zichermann y Christopher Cunningham (2011), la gamificación consiste en el uso de mecánicas, elementos y técnicas de diseño de juegos en contexto que no son juegos para involucrar a los usuarios y resolver problemas [1].

Así pues, el objetivo principal de la gamificación es potenciar las habilidades del aprendiz a través de la dinámica del juego, puesto que mediante la experiencia y la diversión se cree más fácil alcanzar aprendizajes más significativos y funcionales.

Esto es aplicable a varios ámbitos en la vida real. El más usado hoy en día es el **educacional**, donde se introducen técnicas basadas en juegos para potenciar el rendimiento de aprendizaje del alumno. Además del ámbito educacional, la gamificación también es utilizada en el ámbito profesional como por ejemplo en la organización de cursos internos de una empresa o incluso en el ámbito deportivo, donde es común el uso de juegos y competiciones para mejorar una técnica física concreta.

Una de las claves que sustenta la aparición y cada vez más frecuente utilización de estas metodologías de aprendizaje es la forma en la que el ser humano aprende. Si se analiza el *cono de aprendizaje de Edgar Dale*[1] [2], mostrado en la **Figura 1.1** (extraída de [3]), se puede observar que el mayor porcentaje de aprendizaje se logra con un sujeto activo.

[1] Edgar Dale (27 abril 1900 – 8 marzo 1985) fue un pedagogo estadounidense conocido por elaborar el Cono de la Experiencia. Hizo diversas contribuciones en la instrucción visual y auditiva, incluyendo una metodología para analizar el contenido de películas.

Figura 1.1. Adaptación del cono de aprendizaje de Edgar Dale

Para lograr el aprendizaje con sujeto activo existen varios métodos, uno de ellos es a través del juego, puesto que en definitiva, un juego no es más que un conjunto de conductas que, tanto al ser humano como cualquier tipo de animal, les sirve para adquirir y moldear las habilidades necesarias para muchas de las actividades que realizarán a lo largo de su vida.

1.2. Estado del arte

El cambio continuo de la sociedad hace que se necesiten nuevas metodologías de aprendizaje más adaptadas al contexto actual que no las ya conocidas técnicas tradicionales. En un entorno caracterizado por la enorme y continuada evolución de las tecnologías de la información, esta renovación viene liderada de la mano de la gamificación.

Un ejemplo claro de gamificación es el *Colegio Marista "La Inmaculada"* de Granada. Durante el curso 2014-2015, se llevó a cabo una investigación con alumnos de 4º de ESO que escogieron como optativa Física y Química. El proyecto se basó en el empleo de diversas estrategias de gamificación aplicadas a la propia asignatura. Se escogió esa temática por los beneficios pedagógicos de los juegos, por la atracción que ejerce sobre los estudiantes y por el desarrollo de su automotivación. Los alumnos participaron de manera individual, por parejas y por equipos en pruebas como *"Fórmulas químicas a la carrera"* o *"La ruleta de la física y la química"*. Según los resultados publicados, la experiencia fue un éxito ya que observó un incremento del rendimiento académico de la asignatura y los alumnos evaluaron la experiencia en la asignatura como muy positiva [4].

Cabe destacar que actualmente existen algunos ejemplos de aplicaciones desarrolladas que son utilizadas para gamificar el aprendizaje. Algunos ejemplos son los que se muestran a continuación:

- ***El Plan del Héroe***: es un juego de tablero muy útil en la creación de empresas. Se cataloga como toda una aventura de héroes y villanos que guían paso a paso hasta diseñar todos los elementos necesarios para construir la mejor propuesta, algo que solo se consigue aplicando efectivas técnicas de estrategia empresarial [5].

- ***ClassDojo***: proporciona un sistema fácil y rápido al profesorado para la asignación de puntos positivos y negativos en función del comportamiento del alumnado. Permite pues hacer un seguimiento del alumno y compartir el mismo con otros usuarios como por ejemplo padres o tutores legales a través de un ordenador, tableta o Smartphone [6].

- ***Ribbon Hero 2***: es un juego lanzado por Microsoft para facilitar el aprendizaje de su paquete MS Office. Para ello, Microsoft pone al servicio del usuario la mecánica y dinámica de juego [7].

- ***Kahoot!***: consiste en una aplicación docente en la que el profesor puede crear cuestionarios para evaluar a sus alumnos de forma individual o por equipos. Los ganadores serian aquellos que obtienen una mayor puntuación. Además, tiene una gran cantidad de funcionalidades como pueden ser modificar el tiempo de cuenta atrás, compartir con los demás profesores los cuestionarios ya creados o añadir fotos o vídeos [8].

- ***Juego de la Paz Mundial***: se basa en una simulación política que sumerge a los estudiantes en un mundo no muy diferente del que se puede leer en los periódicos. Su objetivo es trabajar valores como la cooperación, solidaridad, creatividad, trabajo en equipo, pensamiento crítico y saber escuchar [9].

- ***Zombie-Based Learning***: es un juego que ayuda a mejorar el conocimiento de la geografía. Se basa en una situación de apocalipsis mundial en el cual el usuario debe escapar. La única manera de huir de los seres apocalípticos es conocer perfectamente la geografía de la zona [10].

- ***Socrative***: se trata de otra herramienta de participación en el aula con la que se trabaja en ordenadores, laptops, tablets y móviles. Destaca por sus cuestionaros simples, con tiempo o con ranking de resultados. Los estudiantes responden en tiempo real a través de sus dispositivos y se generan informes de sus resultados que puedes exportar como Excel [11].

De entre todas estas aplicaciones, ClassDojo, Kahoot! y Socrative son las más similares a **Classpip**, la aplicación sobre la cual se desarrolla este proyecto. Por tanto, analizando sus posibilidades, se advierte una escasez en el sistema de organización de competiciones. Es posible la creación de concursos de preguntas entre estudiantes, o equipos, que se resuelven en un mismo día, pero no hay nada más donde elegir que sea diferente a este mismo tipo de concurso.

Es por eso que, con la implementación de este nuevo módulo de competiciones en Classpip, se conseguirán grandes ventajas respecto a éstos. Aportará gran

diversidad y posibilidades a la hora de organizar competiciones; se podrá elegir entre distintos tipos de competición y diferentes modalidades (individual y equipos) además de que no tienen por qué finalizar en un mismo día, aportando en los alumnos un deseo constante de superación y esfuerzo personal.

Otro aspecto importante reside en la naturaleza de los duelos entre alumnos, ya que éstos no tendrán que ser sólo mediante preguntas, sino en base a cualquier otra cosa. Con ésto se consigue alentar al estudiante a lograr cualquier cosa que se proponga, como podría ser sacar la nota más alta en un examen.

1.3. Beneficios y riesgos

La gamificación es una nueva técnica que conlleva muchos beneficios, entre los cuales destacan los mostrados a continuación:

- Aprendizaje más significativo, permitiendo mayor retención en la memoria al ser más atractivo para el alumno.

- Activación de la motivación por el aprendizaje.

- Compromiso con el aprendizaje y fidelización o vinculación del estudiante con el contenido y las tareas a realizar.

- Mejora de la competitividad, trabajo en equipo y autonomía.

- Oportunidades para incrementar la diversión en clase, siendo este concepto totalmente compatible con el propósito de aprender.

Gamificar parece, a primera vista, una buena idea; aprender utilizando los elementos de los juegos. Pero la utilización de este método también tiene riesgos que pueden agravarse si no se aplica adecuadamente. Entre ellos se encuentran:

- Factor económico: esta nueva metodología de aprendizaje está comenzando a convertirse en una idea de negocio muy lucrativo. Los beneficios que aportan las aplicaciones de gamificación y la utilización de las mismas por parte de los usuarios hacen que los precios suban de forma continuada.

- Consecución de premios: un aspecto que suele ser problemático por lo que se refiere a la gamificación es reducir la metodología a la mera consecución de premios. Esto puede ser un riesgo en el enfoque final del alumno, el cual puede obsesionarse con el premio dejando en segundo término el objetivo principal.

- Linealidad en los juegos: la mayoría de juegos comercializados son de carácter lineal. Esta linealidad no da cabida a la educación personalizada ni a diferentes niveles, intereses y estilos de aprendizaje.

CAPÍTULO 2. PROYECTO BASE Y SUS TECNOLOGÍAS

Para entender el desarrollo y objetivo de este proyecto, que consiste en la realización de un generador de competiciones, es esencial explicar en qué consiste el proyecto original del cual se parte y las bases de las que consta, analizar la arquitectura software y comentar las necesidades para completarlo. Todo esto se detallará en este segundo capítulo.

2.1. Introducción al proyecto original y sus bases

El proyecto descrito en este documento forma parte de una línea de proyectos destinada a implementar una herramienta para la gamificación de la docencia, es decir, el objetivo es crear una aplicación denominada **Classpip** que pueda ser usada a través de diferentes plataformas (móviles, ordenadores, etc.) y que permita a los profesores poder impartir sus clases con este tipo de metodología. Esta línea de proyectos se desarrolla sobre una **plataforma de software de código abierto**, el proyecto base, creada por un alumno de la Universidad Politécnica de Cataluña en 2016 para su proyecto final de máster.

Para cumplir con este objetivo, es necesario conocer la plataforma sobre la cual se irán incorporando los diferentes proyectos que se desarrollen. Por lo tanto, se comentará lo esencial para poder comprender el resto del documento pero si se deseara profundizar más para conocer mejor este entorno de desarrollo creado, se puede descargar la memoria base, donde se detalla en profundidad, desde la web https://upcommons.upc.edu/handle/2117/101237.

2.1.1. Arquitectura del proyecto

Primeramente se debe comentar la arquitectura general, la cual se compone de cuatro elementos: una aplicación **móvil**, dos aplicaciones de escritorio (**panel de administración** y **web pública**) y una **arquitectura orientada a servicios**, **Figura 2.1** (extraída de la memoria mencionada).

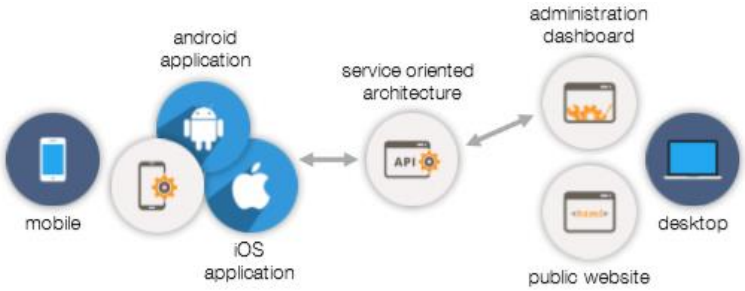

Figura 2.1. Arquitectura del proyecto base

El panel de administración, consiste en una aplicación web que podría ser usada por el administrador de la escuela, el profesor o incluso el alumno para realizar operaciones más complejas como podría ser la configuración o análisis de datos

a partir de informes. Aunque no es necesario que sean complejas, también se pueden realizar otras tareas más sencillas desde él, como la consulta de los grupos de los que se forma parte. La aplicación móvil en cambio, está pensada para realizar tareas sencillas, como realizar alguna actividad de aula. Estas dos piezas comparten datos entre ellas por lo que debe haber un punto común que almacene y exponga los datos, es el caso de la arquitectura orientada a servicios. Ésta se encarga de tramitar todas las peticiones del usuario a través de cualquier frontal (web, dispositivos Android, iOS, etc.). Además, por último existe un sitio web público para fines publicitarios y de marketing.

2.1.2. Modelos existentes

Es necesario también conocer los modelos que componen esta aplicación a su inicio, los cuales son fundamentales en una versión básica. Estos son los siguientes:

- Administrador de la escuela, Profesor y Estudiante: Corresponden a los usuarios que forman parte de la aplicación.

- Escuela, Clase, Curso, Asignatura, Medalla y Avatar: Estos modelos son esenciales para la organización de los individuos. El avatar es una imagen identificativa que se asocia a un usuario.

2.1.3. Funcionalidades básicas

Este apartado se centra en la descripción de las funcionalidades iniciales de la aplicación móvil y del panel de administración. Se deja la web pública de lado debido a que no se introducirán cambios en ella a lo largo de este proyecto.

2.1.3.1. Panel de administración

El panel de administración consta de dos secciones principales:

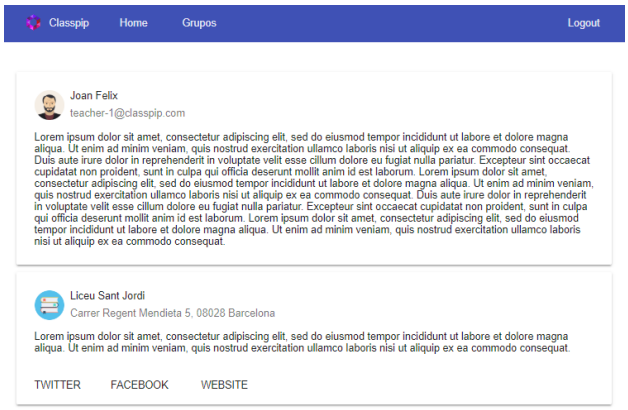

Figura 2.2. Aspecto general (dashboard)

Home: Muestra la información del usuario que haya iniciado sesión. Además, si se trata de un profesor, le dará información acerca de su escuela y le permitirá el acceso al Twitter / Facebook / Web de ésta.

Grupos: Esta sección muestra una lista de los grupos a los que pertenece el profesor o alumno que haya iniciado sesión. Esta lista muestra el nombre de cada grupo junto con el curso y asignatura que lo componen (clase), como puede ser 1º ESO de Inglés.

2.1.3.2. Aplicación móvil

Las funcionalidades de la aplicación móvil varían ligeramente dependiendo de quién inicie sesión (profesor, alumno o administrador de la escuela).

Al iniciar la sesión, todos podrán ver en la sección principal información acerca de la escuela a la que pertenecen y acceder a su localización en el mapa, pero existen algunas diferencias. El administrador de la escuela tendrá la opción de visitar el Facebook / Twitter / Web de ésta y de consultar la lista de profesores y alumnos que pertenecen a ella y así poder entrar en el perfil personal de cada uno. A un profesor en cambio, le aparece una lista de todos sus grupos y puede consultar a través de éstos los alumnos que lo forman.

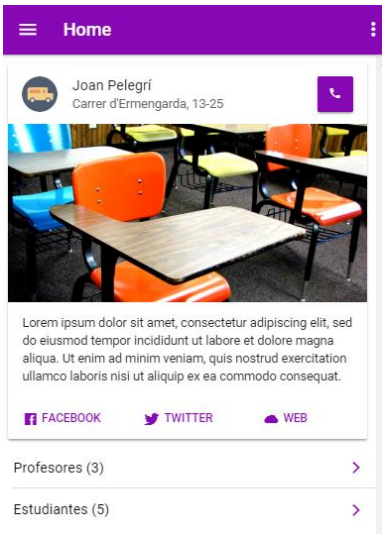

Figura 2.3. Aspecto general (móvil)

Desde el menú, todos pueden acceder a su perfil personal, al mapa de su escuela, volver a la página principal o cerrar sesión.

2.2. Arquitectura software

Habiendo realizado una introducción básica para tener una idea de en qué consiste la aplicación y de lo que la constituye, es momento de hablar sobre las **tecnologías** que la sustentan. Para ello, este apartado se centrará en profundizar sobre cada uno de los fragmentos en los que se divide la arquitectura, ya mencionada en el subapartado 2.1.1, comentando la pila de tecnologías que los componen.

Cabe decir que todas estas tecnologías han sido las utilizadas para implementar la aplicación a su inicio y por tanto, con el paso del tiempo, es lógico que se vayan desarrollando herramientas interesantes para añadir, o que se deban de actualizar algunas de las tecnologías ya incluidas hacia versiones más recientes.

La **Figura 2.4** consiste en un diagrama extraído de la memoria de Classpip debido a la claridad con la que expone y resume todas las herramientas y frameworks[2] [12] empleados en cada componente del software.

2.2.1. Arquitectura orientada a servicios

La arquitectura orientada a servicios es un componente que cuenta con los datos y la lógica necesaria para actuar como un único back-end que puede ser utilizado

[2] Framework, o marco de trabajo, podría definirse, en el contexto de desarrollo de software, como el esquema conceptual y tecnológico que puede ayudar al desarrollo y/o implementación de una aplicación.

en los diferentes frontales (escritorio y móvil) para gestionar las peticiones de estos. Se trata de una **API** (Interfaz de programación de aplicaciones) [13] REST[3] [14] que ha sido creada con una herramienta software de código abierto de IBM llamado **StrongLoop** [15]. Se encuentra configurada en lenguaje **JavaScript** [16] y expuesta a través de una infraestructura web llamada **Express** [17] en el entorno **Node.js** [18]. La interfaz de programación de aplicaciones almacena la información en una base de datos **MySQL** [19].

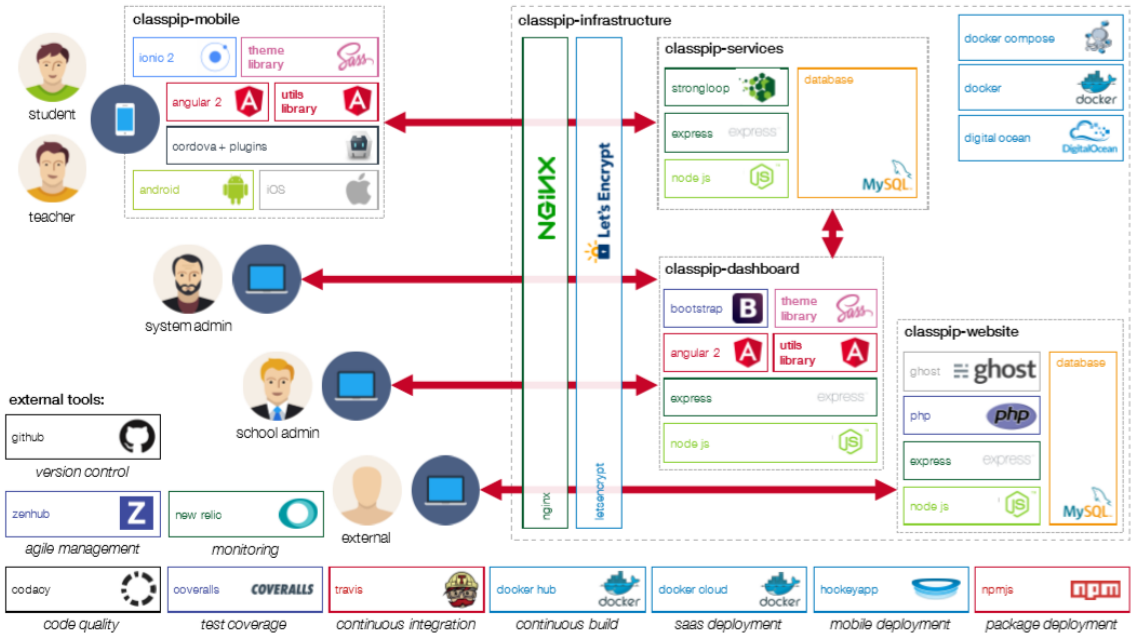

Figura 2.4. Arquitectura software

2.2.2. Panel de administración

El panel de administración, o *dashboard*, está pensado para ser utilizado a través del escritorio, aunque podría ser accesible también desde el móvil. Por tanto, este se creó con **Angular 2** [20], desarrollado en **TypeScript** [21], el cual nos permite poder reutilizar paquetes o código para cualquier plataforma (web, web móvil, escritorio, etc.), acompañado por un kit de herramientas de código abierto llamado **Bootstrap** [22], que aporta el estilo en la web para la interfaz del usuario.

Actualmente, Angular se encuentra en la versión 4 y además, se ha introducido un componente llamado **Angular Material** [23] para el diseño de la interfaz del usuario.

Por último, para visualizar e interactuar con el dashboard, éste se expone a través del servidor web **Express** en el entorno **Node.js**, al igual que la API.

[3] La transferencia de estado representacional (o REST, *REpresentational State Transfer*), es un estilo de arquitectura software que destaca por no tener estado, es decir, un usuario debe acceder a la aplicación a través de su usuario y clave para que el servidor le devuelva un token con el que deberá de acompañar todas sus peticiones para poder ser identificado por el servidor.

2.2.3. Aplicación móvil

La aplicación móvil se ha creado con **Ionic 2** [24], que se trata de un framework basado en **Angular 2** para el desarrollo de aplicaciones híbridas. Además, se hace uso de **Apache Cordova** [25], que consiste en un framework de desarrollo móvil de código abierto que permite usar tecnologías web estándar (HTML5, JavaScript, etc.) para el desarrollo multiplataforma, y por tanto, permite la conversión de la plataforma nativa a **Android** e **iOS**, evitando así tener que crear código exclusivo para cada una de estas. También es necesario incluir algunos plugins para acceder a algunas funciones del móvil.

Por último, para el diseño de la interfaz del cliente, se utiliza la librería de hojas de estilo **Sass** [26].

2.2.4. Herramientas externas

Una vez comentadas las tecnologías utilizadas en cada uno de los componentes de la arquitectura, es momento de explicar las herramientas externas que también forman parte de esta aplicación.

Primeramente, para conseguir un desarrollo colaborativo, todos los componentes de la aplicación se alojan en repositorios de una plataforma llamada **Github** [27], desde la cual es posible descargarlos, añadir y/o modificar parte del código y volver a guardarlos en ésta con los nuevos cambios. Una herramienta complementaria a Github es **ZenHub** [28], que permite crear tableros y realizar un seguimiento de las tareas, mejoras y errores en los proyectos pudiendo conseguir un desarrollo ágil de software.

Para verificar la cobertura, el estilo, la seguridad, la duplicación, la complejidad y la calidad del código durante del desarrollo de éste y así conseguir un mejor entorno de integración continua[4] [29], se utilizan **Coveralls** [30] y **Codacy** [31]. Aunque **Travis CI** [32], es la herramienta decisiva para comprobar el correcto desarrollo del código y su posible integración en el código fuente inicial a través de *Pull Requests*. Se trata de un servidor de integración continua que realiza una serie de pruebas contenidas en un fichero de configuración del proyecto para verificar el correcto funcionamiento y permitir su construcción.

Después del desarrollo, testeo y validación de la aplicación, solo queda su construcción y su distribución. Por tanto, para la web se realiza a través de **Docker** [33], que automatiza el despliegue dentro de contenedores y se alojan en el servidor **DigitalOcean** [34]. Para la aplicación móvil es diferente, ésta se construye localmente a través de Ionic y Cordova y es distribuida en su versión beta para recoger el feedback a través de **HockeyApp** [35].

El gestor de paquetes javascript **npm** [36] permite instalar, compartir y distribuir código además de gestionar las dependencias en los proyectos.

[4] Por integración continua se entiende la compilación y ejecución de pruebas a lo largo del desarrollo del software para poder detectar los fallos de forma temprana.

Finalmente, con **New Relic** [37] es posible monitorizar la aplicación, es decir, es capaz de realizar tareas como pueden ser la monitorización de las conexiones HTTP (tiempos de respuesta, etc.) o de los errores entre otras muchas.

2.3. Necesidades iniciales y proyectos paralelos

Como se ha dicho en el apartado 2.1, existe una línea de proyectos que tienen un fin: completar *Classpip* para que pronto pueda ser una aplicación completa y funcional destinada a la gamificación de entornos docentes. Para ello, hay unos requisitos esenciales que están siendo cubiertos por diferentes proyectos:

- **Módulo de puntos e insignias:** Consiste en un sistema para otorgar puntos o insignias a los estudiantes por parte del profesor.

- **Módulo de colecciones de cartas**: Se trata de una sección en la que los profesores crean conjuntos de cartas para que los alumnos coleccionen.

Estos dos módulos cubren la necesidad de motivar al alumnado con premios como recompensa. Consiguen que los alumnos deseen obtener un gran número de puntos y de insignias y ganar todas las cartas existentes de cada colección.

- **Módulo de preguntas de tipo test**: se trata de un generador de preguntas de tipo test para resolver por parte de los alumnos.

- **Módulo de gestión de equipos**: Es una herramienta que permite crear y gestionar equipos de estudiantes para realizar cualquier tipo de actividad.

2.3.1. Generador de competiciones

El objetivo de este proyecto es la creación de una herramienta que consiste en un **generador de competiciones** que se gestiona a través del **panel de administración**. Además, también se añadirán otros módulos: el de creación de equipos (de forma provisional) y algunas funcionalidades en la aplicación móvil.

Las competiciones están formadas por un número determinado de jornadas en las cuales se producirán los enfrentamientos entre los alumnos con el fin de ganar. Podría integrarse perfectamente con los demás módulos para hacerlo más interesante. Un ejemplo sería la creación de una competición en la que los enfrentamientos de alguna jornada, se deciden mediante un concurso de responder correctamente preguntas por equipos, y una vez haya finalizado la competición, a los ganadores se les podría asignar puntos u otorgar cartas o insignias como recompensa por su victoria.

Cuando la herramienta que se va a desarrollar en este proyecto esté terminada, se podrán crear las competiciones, gestionarlas durante su transcurso y eliminarlas una vez finalizadas. Todas las funcionalidades que incluye la aplicación serán explicadas con detalle en los siguientes capítulos.

CAPÍTULO 3. COMPETICIONES

Este capítulo es de suma importancia para entender cuál es el principal objetivo de este proyecto. Explica las bases del tipo de competiciones que se van a poder crear en la aplicación y la apariencia y funcionalidades que tendrán en el **panel de administración**.

3.1. Bases de la competición y sus tipos

Una competición es una colección de enfrentamientos entre jugadores que puede desarrollarse en una o varias jornadas. Se organiza siguiendo unas reglas determinadas y da como resultado una ordenación de los jugadores o como mínimo un ganador.

Los enfrentamientos de cada jornada pueden dilucidarse a partir de lo que decida el profesor a lo largo de la competición, es decir, al crear una competición con la herramienta que se ha desarrollado, no se generará una competición en la cual se encasillen los enfrentamientos y el ganador sea quien acierte más preguntas en todas las rondas por ejemplo, sino que la naturaleza de estos puede ser decidida por el profesor en cualquier momento y no viene determinado por la competición ya desde su inicio.

Poniendo un ejemplo para el párrafo anterior, los ganadores de la primera jornada pueden decidirse a través de los resultados de un examen que se ha hecho en clase, para la segunda jornada, se podría hacer uso del componente de preguntas de respuesta múltiple y que ganara quienes aciertes más, otra jornada podría decidirse por resolver en menos tiempo un problema propuesto en clase, y así hasta finalizar todas las jornadas. Incluso en alguna se podría elegir de forma aleatoria al ganador, introduciendo así un elemento de azar que puede hacer la competición más interesante.

Este proyecto se centra en la creación de dos tipos diferentes de competición, **tipo liga** y **tipo torneo de tenis de doble eliminación**, en los que se podrá participar de forma individual o a modo de equipo.

Son dos estilos de competición muy comunes con normas bastante sencillas donde los estudiantes tendrán una participación elevada, sobre todo en la de tipo liga, y además, mantendrán el espíritu competitivo siempre activo ya que son competiciones donde el ganador no se conoce en un mismo día sino al cabo de un tiempo. Por tanto, esto alienta al alumno a querer mejorar para poder escalar de posición en caso de ser una competición de tipo liga, o a querer llegar a la final en caso de ser la de tipo torneo de tenis.

3.2. Sección de competiciones

Para llevar a cabo todas las acciones relacionadas con las competiciones dentro de la aplicación, es necesario la creación de un apartado dedicado a éstas. Este

apartado es la sección de competiciones, que actúa como un menú principal desde el cual se puede acceder a las competiciones existentes y además, en caso de ser profesor, crear nuevas.

Dentro de la web de Classpip, se utiliza una **barra de navegación** para navegar entre las distintas secciones. Por esta razón, se accede a la sección de competiciones desde ésta.

Figura 3.1. Sección de competiciones para profesores.

En la **Figura 3.1** se puede observar este menú principal para los profesores. Puede darse la situación de no tener competiciones o que se inicie sesión como estudiante, por tanto, en el **anexo 9.1.1** puedes encontrar las diferentes apariencias dependiendo del caso.

3.2.1. Crear nueva competición

La creación de una competición, te dirige a una nueva sección donde habrá que seguir una serie de pasos, pero antes, habrá que escoger el tipo de competición que se desea realizar, **Figura 3.2**.

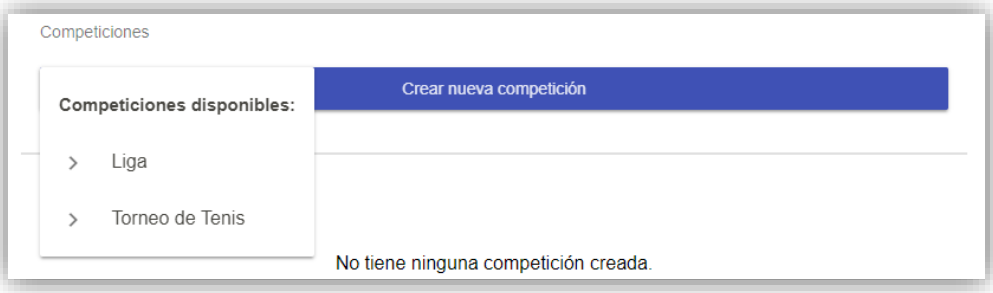

Figura 3.2. Crear nueva competición (elección del tipo)

Cada competición tendría que seguir una serie de pasos diferentes dependiendo de los parámetros de entrada necesarios para su creación. En el caso de estas dos competiciones solo divergen en que la liga requiere un parámetro más en el primer paso, el número de jornadas.

3.2.1.1. Primer paso: Inicialización

En primer lugar, el profesor debe introducir los parámetros que son obligatorios para la **inicialización**. Se trata de determinar el nombre de la competición, la modalidad (individual o por equipos), la clase que participará y el número de jornadas para el tipo liga. Este proceso puede observarse en la **Figura 3.3**.

Figura 3.3. Creación competición tipo liga (paso 1)

3.2.1.2. Segundo paso: Participantes

Es la hora de seleccionar a los jugadores que formarán parte de la competición ya que es posible que no se desee que participe la clase entera.

En el paso anterior hubo que elegir la clase y el modo, esencial para este paso. Si se trata de una competición individual aparecerá una lista de los estudiantes pertenecientes a la clase elegida y en caso de haber seleccionado el modo equipo, la lista será de los equipos de esa clase, que han tenido que ser previamente formados en otra sección de la aplicación.

El aspecto de este segundo paso para modo individual se muestra en la **Figura 3.4**, donde aparece el nombre y apellido de cada alumno. Para los equipos aparecería el nombre de éstos.

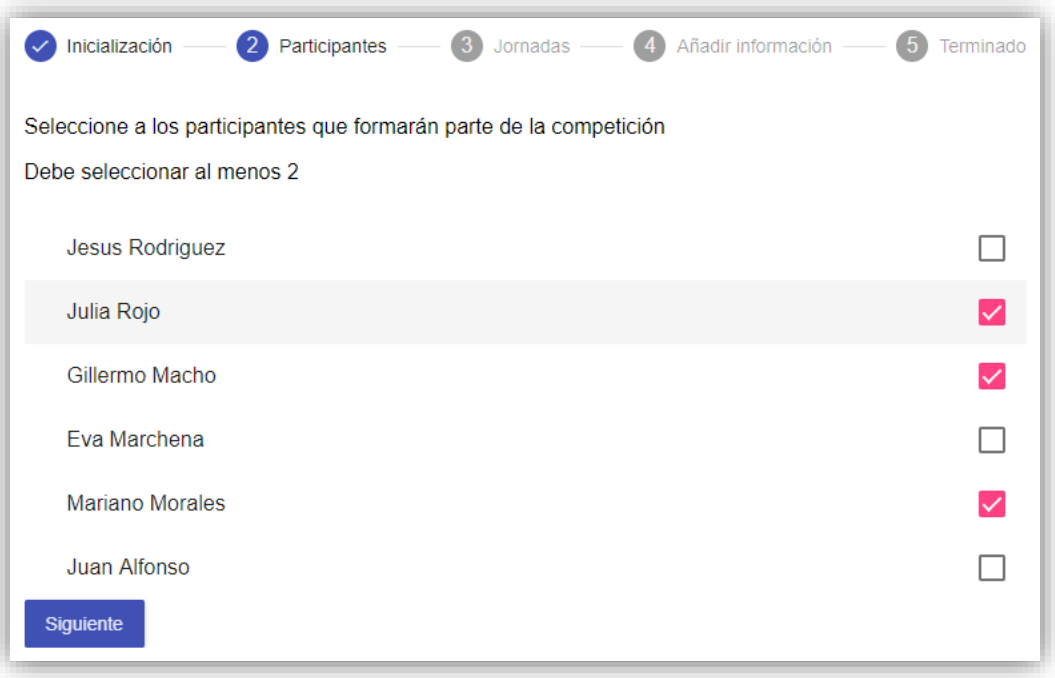

Figura 3.4. Creación de la competición (paso 2)

3.2.1.3. Tercer paso: Jornadas

Una vez configurada la parte obligatoria (primer y segundo paso), lo siguiente es introducir las fechas de las jornadas mediante un calendario que se despliega sobre la jornada seleccionada. Véase **Figura 3.5**.

El número de jornadas que aparecen es el introducido anteriormente o si se trata de un torneo de tenis es la propia aplicación la encargada de calcular este número automáticamente.

Es posible no saber en qué fecha tendrá lugar cada jornada y que se prefiera especificar en otro momento, por tanto, este paso es opcional.

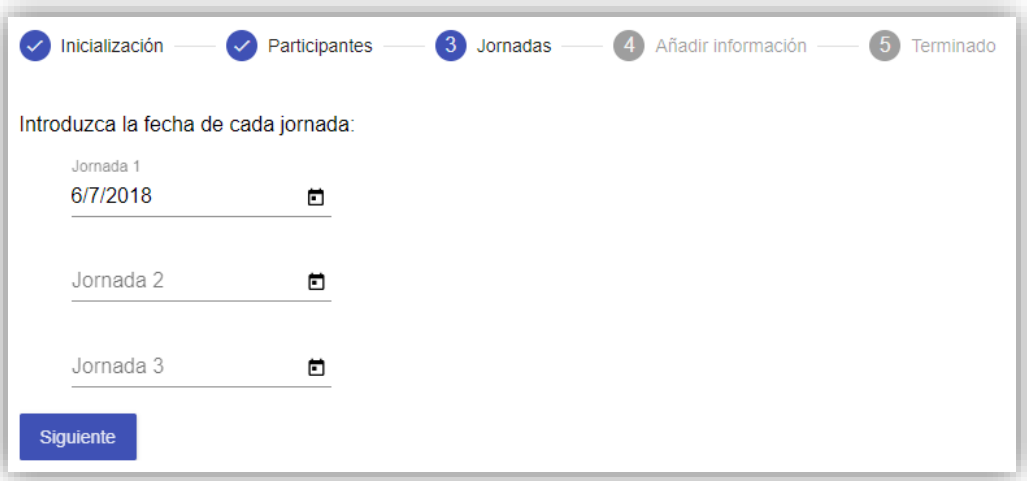

Figura 3.5. Creación de la competición (paso 3)

3.2.1.4. Cuarto paso: Añadir información

Con las fechas de las jornadas ya establecidas, lo siguiente que hará el profesor será añadir un breve texto informativo sobre la competición. En él, puede escribir en base a qué se decidirán los resultados de cada jornada o cualquier aclaración que crea conveniente. Véase **Figura 3.6**.

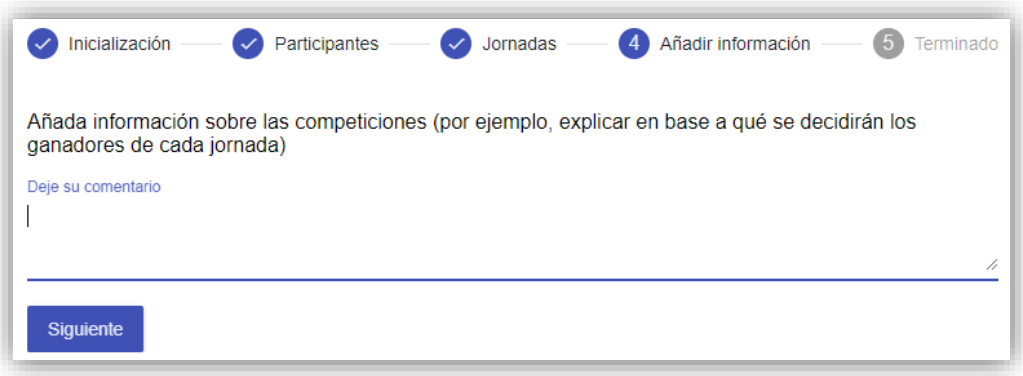

Figura 3.6. Creación de la competición (paso 4)

3.2.1.5. Quinto paso: Terminado

Por último, una vez completado el cuarto paso, se mostrará un texto informando de que la competición ha sido creada, **Figura 3.7**.

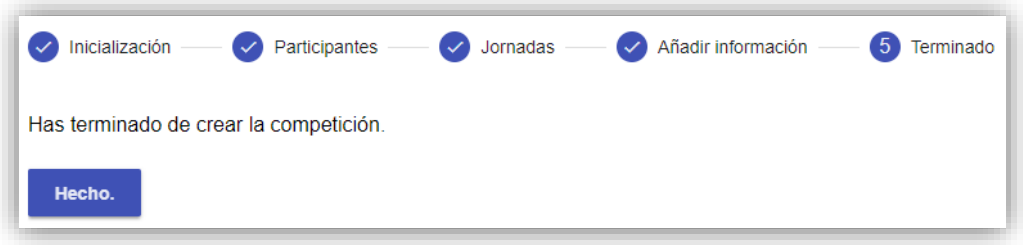

Figura 3.7. Creación de la competición (paso 5)

3.2.1.6. Restricciones

Como se ha visto, el segundo paso depende del primero y por tanto, como se observa en la **Figura 3.8**, si algún campo ha quedado sin completar no se podrá continuar al próximo paso. Te avisa pintando de rojo todos los campos que no han sido completados y deshabilitando el botón de siguiente.

En el paso de participantes tampoco te dejará continuar hasta no haber seleccionado un mínimo de dos.

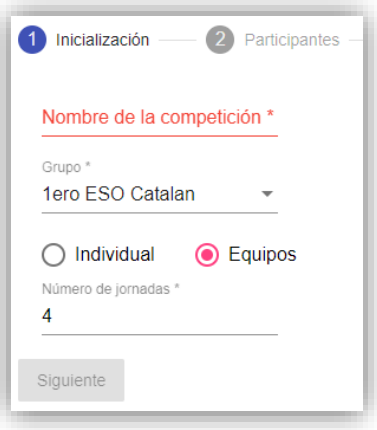

Figura 3.8. Campos incompletos

3.2.2. Acceso a una competición

Para poder distinguir cuál es la competición a la que se quiere acceder, se muestran una serie de características que la determinan:

- El nombre que se le haya asignado a la competición.
- El tipo de ésta (liga o tenis).
- La clase que está participando (curso y asignatura).

Los iconos y colores de cada competición son los que hacen referencia al tipo de competición. Por ejemplo, si se trata de un torneo de tenis, será el color verde y el icono de una copa lo que la caracterizará, **Figura 3.9**.

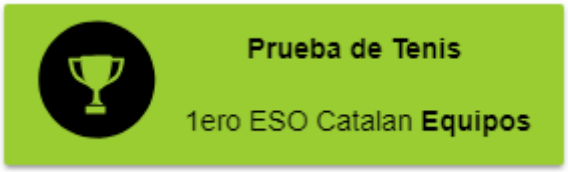

Figura 3.9. Apariencia competición tipo Tenis

Una vez visto el menú principal de competiciones, toca analizar con detalle en qué consisten los tipos de competición creados y cómo lo implementamos.

3.3. Liga

En este apartado se explican las bases de la liga y las funcionalidades que tiene en la aplicación, además de añadir figuras sobre el aspecto de ésta para facilitar la comprensión.

Debido a la gran cantidad de diferencias de aspecto a causa de acceder como profesor o alumno, individual o equipos, competición completa, etc., en los **anexos 9.1.3** y **9.1.5** se encuentran recogidas todas las posibles apariencias por muy pequeñas que sean estas desigualdades para el que desee contemplarlas.

3.3.1. Bases de la liga

En este tipo de competición, los jugadores se enfrentan todos contra todos en batallas individuales con el objetivo de ganar puntos y ocupar un buen lugar en la clasificación general de la clase. Si se jugara con la modalidad de equipo, se enfrentarían todos los equipos contra todos.

Respecto al sistema de puntuación, los ganadores de los duelos obtienen tres puntos, los perdedores no consiguen sumar ninguno y en caso de empate, ambos alumnos reciben un punto.

Puede darse el caso de que el número de participantes o de equipos sea impar, y por tanto, todos los jugadores o equipos deberían descansar una jornada diferente, en la cual no sumarían puntos.

Finalmente, los resultados de los enfrentamientos de cada jornada se pueden decidir en base a diferentes criterios, que pueden ser determinados en el momento de crear la competición por parte del profesor.

3.3.2. Su apariencia en el panel de administración y sus funcionalidades

Cuando accedemos a alguna competición de tipo liga desde la sección de competiciones, navegaremos a una nueva sección que mostrará notables diferencias dependiendo de si eres alumno o profesor.

En ambos casos, tendrán la posibilidad de consultar las jornadas, la clasificación, la información que introdujo el profesor al crear la competición y los miembros que componen cada equipo si los hubiera.

Un profesor, además, podrá modificar las fechas de cada jornada, corregir el texto explicativo que escribió sobre la competición, introducir los resultados de los enfrentamientos y eliminar la competición si así lo desea, véase **Figura 3.10**.

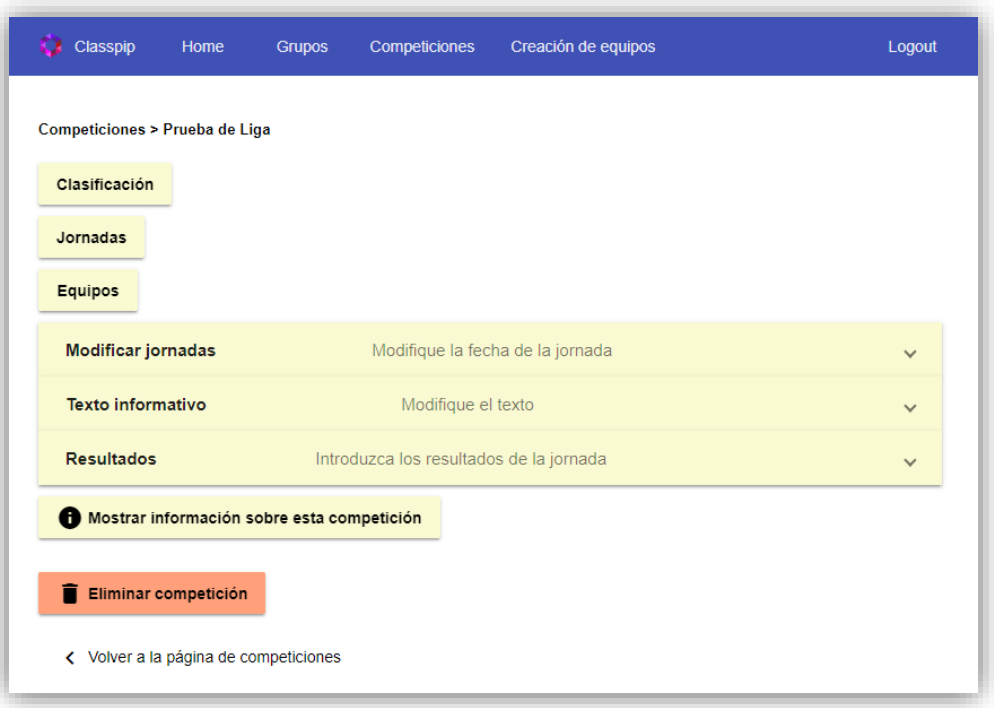

Figura 3.10. Menú principal de liga para profesores

En la sección de los alumnos en cambio, la única función diferente añadida será la de empezar el enfrentamiento de la jornada en caso de que éste tuviera que realizarse mediante la aplicación, aunque en este proyecto no se le dará ninguna funcionalidad.

3.3.2.1. Clasificación

La clasificación, como se observa en la **Figura 3.11**, muestra el puesto que ocupa cada alumno o equipo (equipo en este caso) según los puntos obtenidos. Además, aporta otros datos interesantes como son el número de partidas jugadas, ganadas, perdidas y empatadas. En caso de haber un empate (mismos puntos), no hay ningún criterio para ocupar una posición superior, es aleatorio.

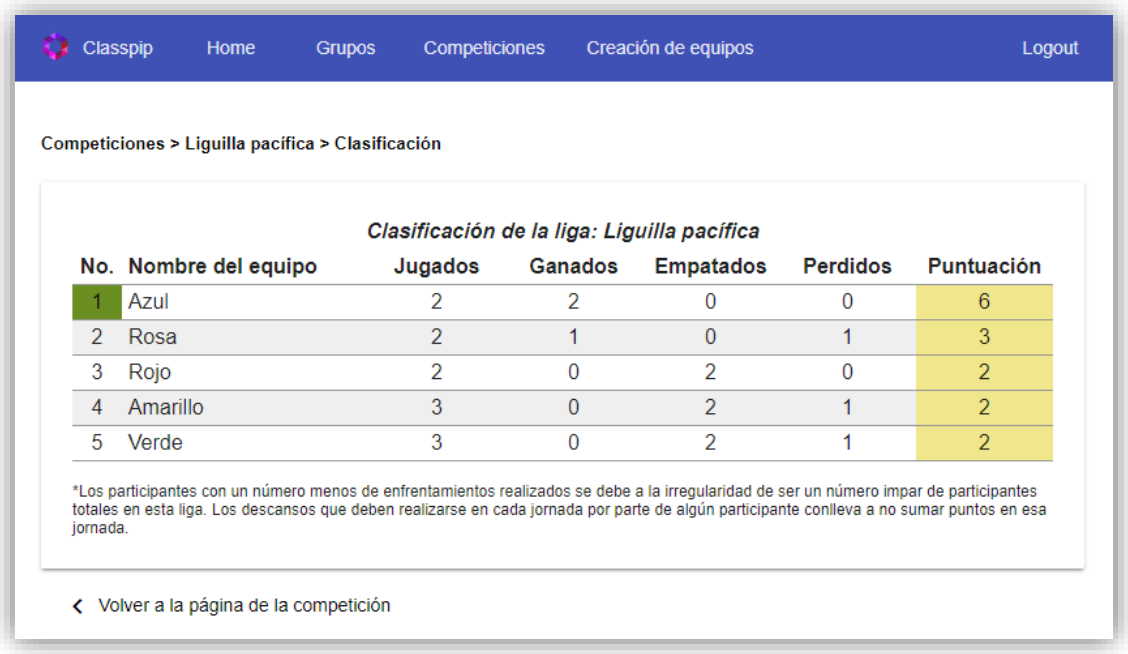

Figura 3.11. Clasificación de la liga (modo equipo y número impar)

3.3.2.2. Jornadas

Figura 3.12. Calendario de la liga por equipos

La aplicación genera los emparejamientos de cada jornada automáticamente en el momento de crear la competición. Gracias a esto y a las fechas que el profesor ha debido de introducir, se puede generar una página de consulta de todas las jornadas, como la que se muestra en la **Figura 3.12**.

En ésta se exponen las fechas y los emparejamientos de cada jornada, y en el caso de que ya se hayan realizado los duelos, los ganadores.

El objetivo de esta sección es ayudar tanto al profesor y al alumno a llevar un seguimiento de los resultados de cada uno de los enfrentamientos y sobre todo, a conocer la fecha de la siguiente jornada para prepararse si fuera necesario.

3.3.2.3. Equipos

La sección de equipos es de mera información, la **Figura 3.13** muestra el aspecto de ésta.

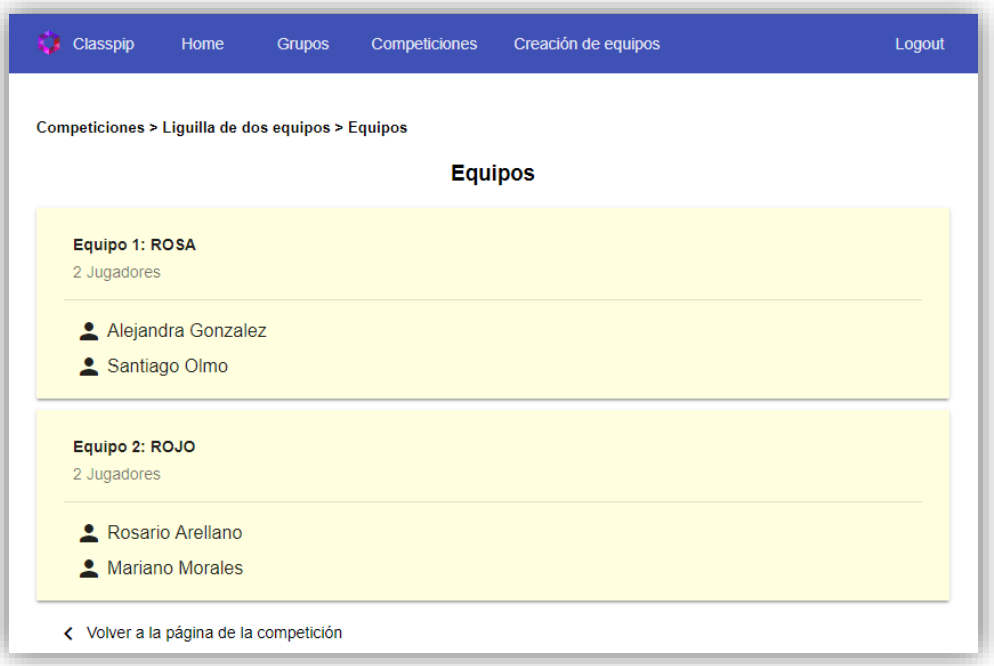

Figura 3.13. Página de equipos

Los alumnos la podrían consultar para conocer a los rivales del equipo contrario mientras que los profesores para recordar los miembros que componen cada equipo.

3.3.2.4. Modificación de fechas

Llevar un control de las fechas de cada jornada desde un principio conlleva conseguir una mejor organización, por eso existe la posibilidad de introducirlas todas al crearse la competición.

También es lógico que los profesores prefieran no introducirlas al comienzo o que por determinadas situaciones o causas fuera necesaria la modificación de alguna de ellas, por tanto, es fundamental incluir esta herramienta.

Ésta consiste en un panel de expansión que contiene dos campos a rellenar, el número de jornada a modificar y su nueva fecha. Solo sería necesario seleccionar la jornada que se desea cambiar y elegir en el calendario que se despliega el nuevo día en el que se efectuará. Puede verse un ejemplo en la **Figura 3.14**. Una vez modificada la fecha, aparece un mensaje temporal informando de que se ha actualizado con éxito.

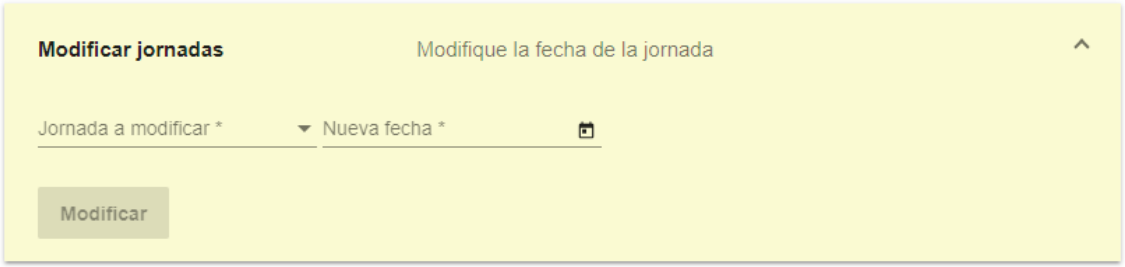

Figura 3.14. Herramienta de modificación de jornadas.

En el momento de seleccionar la jornada que se desea modificar, sólo aparecerán disponibles aquellas cuyos encuentros no se hayan disputado. Es decir, si el profesor ha introducido los resultados de alguna jornada, ésta ya no podrá ser modificada. Y en el caso de que no se pudiese cambiar ninguna fecha, esta herramienta mostraría un mensaje informándolo.

3.3.2.5. Introducción de los resultados

Otra herramienta disponible para el profesor, es la de introducir los resultados de los enfrentamientos. Como se ve en la **Figura 3.15**, se trata también de un panel de expansión, donde se puede escoger introducir los resultados de forma aleatoria o manualmente (en esta figura está seleccionada la opción aleatoria).

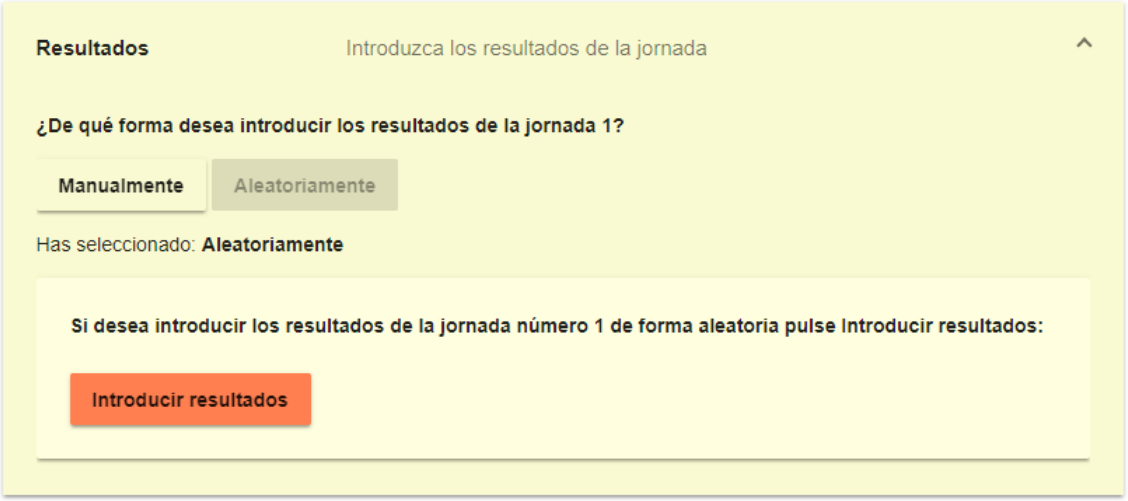

Figura 3.15. Introducción de resultados de forma aleatoria

En caso de haber marcado la opción de manualmente, aparece una lista con los emparejamientos de ese día, véase **Figura 3.16**. Para cada uno de estos duelos hay tres botones de radio con diferentes opciones: en el primero se lee el nombre de una de las personas que se enfrentan en el duelo, en el segundo se muestra la posibilidad de empate y el último es para el otro jugador. Por tanto, hay que seleccionar el nombre del estudiante que sea el ganador del duelo o en caso de empate, seleccionar este segundo botón.

Figura 3.16. Introducción de resultados de forma manual (liga)

Si la competición es por equipos, en vez de aparecer los nombres de los estudiantes, aparecen los nombres de los equipos.

Por último, es necesario añadir que el día que esta herramienta te muestra para que introduzcas los resultados es el de la última jornada que debe jugarse. La propia aplicación es la encargada de mostrártelo de forma automática.

3.3.2.6. Modificación del texto explicativo

El texto informativo que el profesor introdujo al crear la competición, sirve para definir y aclarar conceptos relacionados con ésta, como por ejemplo la naturaleza de los enfrentamientos. Puede suceder, que por cualquier motivo, el profesor quisiera modificar o añadir cualquier cosa, por tanto, es necesario que exista una herramienta para ello.

De nuevo, esta herramienta es un panel de expansión en el que aparece un área de texto donde está escrita la descripción anterior. Para modificarlo, simplemente hay que añadir o alterar lo que se desee y aceptar este cambio. Puede observarse en la **Figura 3.17**.

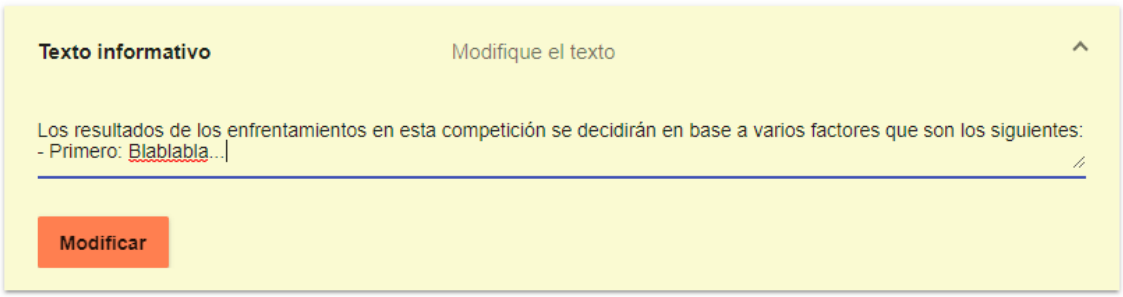

Figura 3.17. Modificación del texto informativo

3.3.2.7. Visualización del texto informativo

Al cargar el menú principal de la liga, este texto se encuentra escondido pero existe la posibilidad de mostrarlo y volver a ocultarlo. En la **Figura 3.18** se puede observar la sección que se muestra al marcar la opción de mostrar.

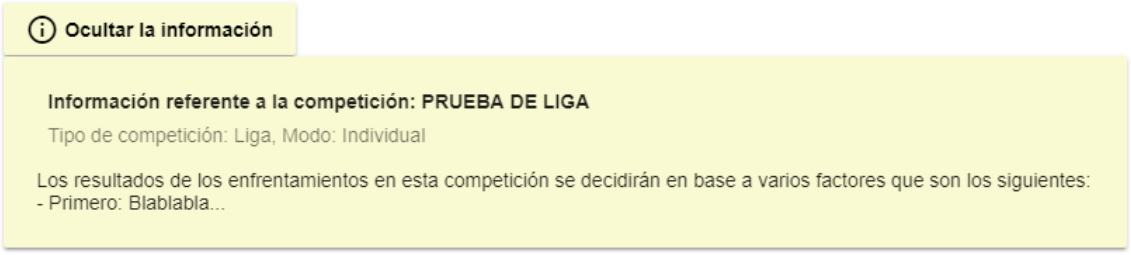

Figura 3.18. Texto informativo desplegado

3.3.2.8. Eliminación de la competición

Por último, es imprescindible añadir la posibilidad de eliminar la competición. El profesor debe de ser capaz de eliminarla una vez finalizada ésta o incluso en cualquier momento si lo creyera necesario.

Al escoger esta opción, la aplicación, para asegurarse de que se desea eliminar la competición, emitirá un mensaje de advertencia que se debe de aceptar para seguir adelante o rechazarlo si ha sido una equivocación (véase **Figura 3.19**).

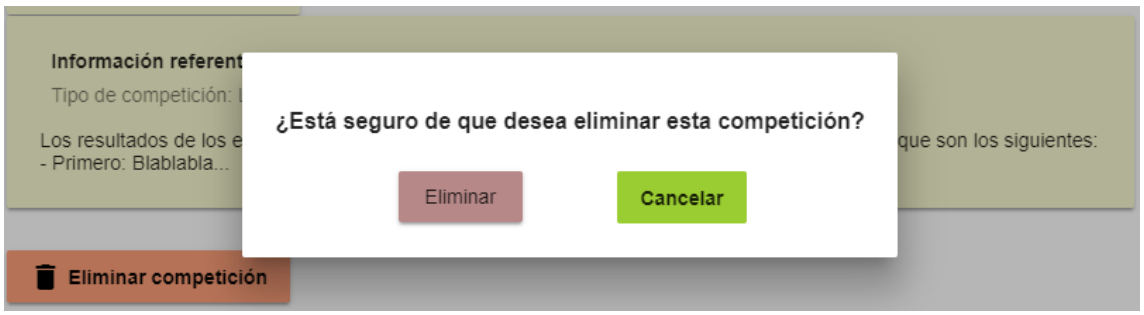

Figura 3.19. Mensaje de advertencia para eliminar una competición

3.4. Torneo de tenis de doble eliminación

En este apartado además de explicar las bases de este tipo de competición, las funcionalidades que tiene en la aplicación y su aspecto, es necesario aclarar la mecánica de emparejamiento y su desarrollo.

Al igual que en la liga, en el **anexo 9.1.4** se encuentra la recopilación de los diferentes aspectos que tiene la aplicación debido a las diferencias que pueden ocasionarse según diversos factores, como pueden ser el número de participantes o su modalidad. También podemos encontrar algunas figuras de esta sección en el **anexo 9.1.5** debido a que no difiere respecto a la liga.

3.4.1. Bases del torneo

Este otro tipo de competición tiene como objetivo ir superando diferentes rondas para llegar a la final, vencer y proclamarse ganador.

Todos los jugadores serán emparejados para enfrentarse entre ellos en una primera ronda y sólo los ganadores de este primer duelo pasan al siguiente. Aunque existe una singularidad con respecto al torneo de tenis clásico, hay doble eliminación. Por tanto, los jugadores que van siendo eliminados, tendrán la oportunidad de llegar a la final compitiendo también por parejas desde un torneo secundario. Una vez pierdan en este torneo auxiliar, quedarán eliminados definitivamente.

Se añade esta peculiaridad para dar otra oportunidad a los perdedores y así incentivarles a esforzarse más para no volver a perder y llegar a la final. Además, de esta forma también habrá más participación en el torneo y será más equilibrado.

Un torneo como éste funciona mejor si el número de jugadores es potencia de dos (4, 8, 16, 32, etc.). Es por eso que si se quisiera crear una competición de este tipo con un número de participantes que no cumplen con este requisito, sería necesario añadir jugadores ficticios (jugadores fantasma). Esto significa la introducción de un elemento de azar, ya que el emparejamiento de cualquier alumno con alguno de éstos significaría un pase directo a la siguiente ronda, es decir, ganaría el duelo de esa jornada.

3.4.2. Mecánica de emparejamiento y desarrollo

Para entender mejor la mecánica de esta competición, en la **Figura 3.20** se pueden observar los emparejamientos de cada ronda y el estilo ramificado que sigue. En este ejemplo, participan 5 alumnos y son añadidos 3 jugadores fantasma para poder llegar a 8, que es el número potencia de dos más cercano.

Para la primera ronda, los emparejamientos se deciden de forma aleatoria, pudiendo coincidir dos fantasmas, en cuyo caso pasa a la siguiente ronda cualquiera de los dos.

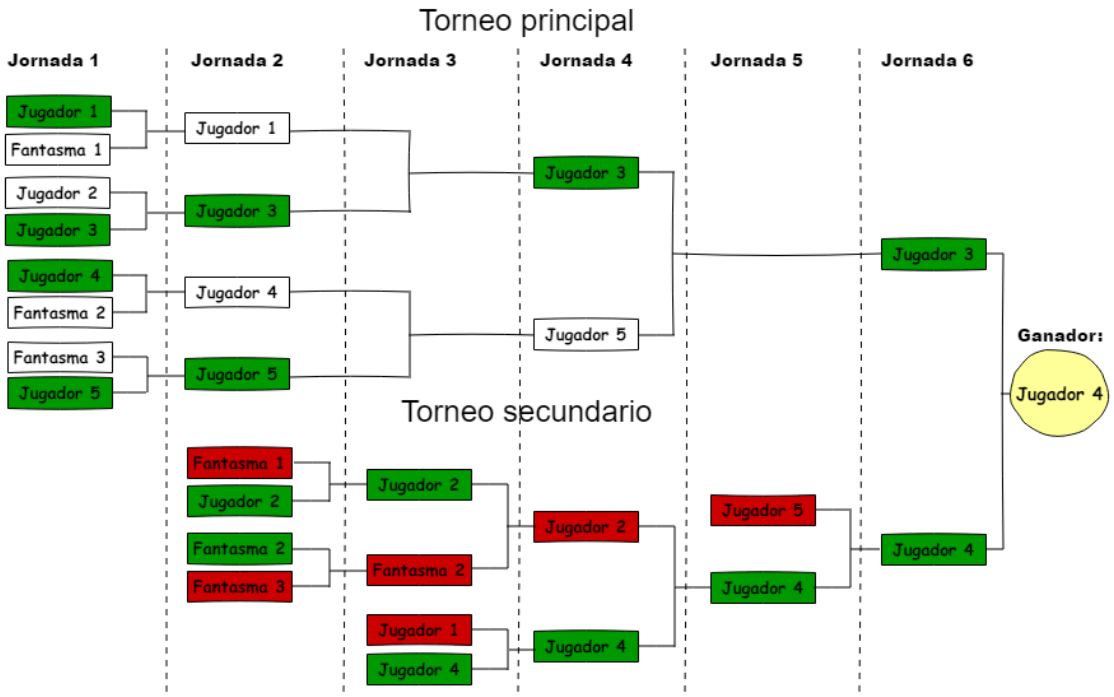

Figura 3.20. Esquema torneo de tenis de doble eliminación para 8 jugadores

Siguiendo con el ejemplo de los 8 participantes, una vez se haya realizado la primera ronda, el número de jugadores se reducirá a la mitad, quedando así 4 en el torneo principal y otros 4 en el secundario. Cuando haya el mismo número de jugadores en ambos torneos, la jornada siguiente la realizan a la vez, como es el caso de la segunda jornada, donde se eliminan dos jugadores de cada torneo.

La siguiente jornada es la tercera. En ésta sólo podrán participar los alumnos del torneo secundario ya que se desigualarán en número debido a que los alumnos que habían perdido en la jornada 2 del torneo principal entrarán en el secundario y volverían a ser cuatro. Una vez jugada esta tercera, en ambos torneos vuelve a haber el mismo número de jugadores, 2, por lo que ambos torneos participarán en la cuarta jornada.

Este es el mecanismo que seguirán los enfrentamientos, eliminándose jugadores en ambos torneos hasta que se enfrenten en la final un jugador del torneo principal y otro del secundario para disputar el puesto de ganador.

Al igual que en la competición de tipo liga, si la modalidad de juego fuera por equipos, en lugar de realizar los emparejamientos entre jugadores, se realizarían entre equipos. Además, también la naturaleza de los enfrentamientos de cada jornada y las fechas de éstas pueden ser determinadas en el momento de crear la competición.

3.4.3. Su apariencia en el panel de administración y sus funcionalidades

Al acceder a una competición existente de este tipo, el aspecto que tiene la sección es parecido al de tipo liga, pero en vez de poder consultar la clasificación, se podrá observar un seguimiento del torneo (véase **Figura 3.21**). También existirán diferencias dependiendo de la modalidad o de si accede un profesor o un alumno.

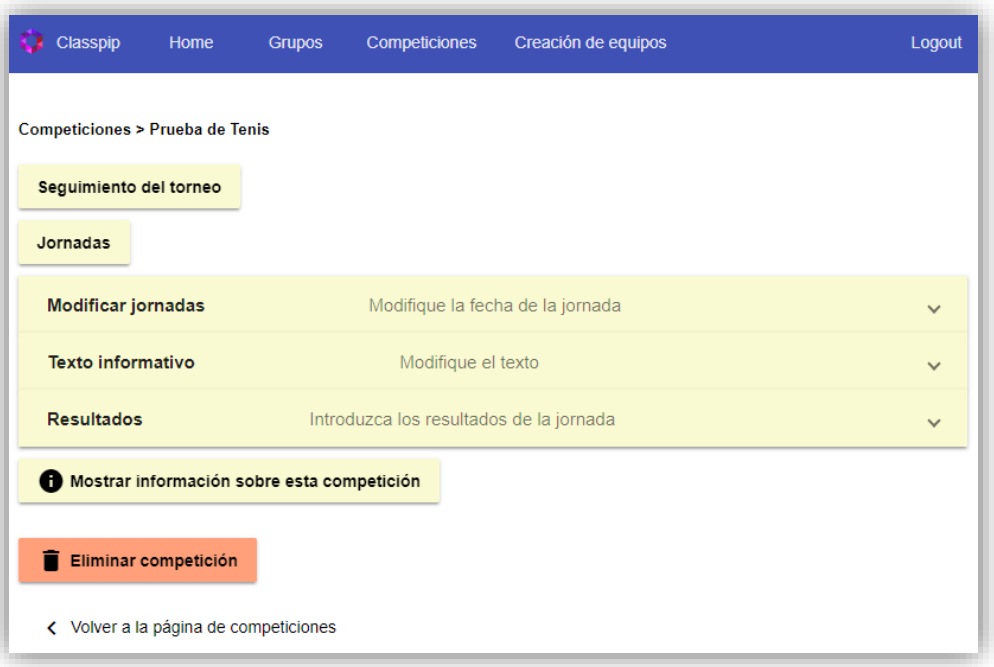

Figura 3.21. Menú principal del torneo de tenis para profesores

Profesor y alumno comparten las secciones de consultar las jornadas, la de equipos en caso de que existan, la de mostrar el texto informativo y por último, el seguimiento del torneo. Este primero además tendrá la opción de modificar las fechas de las jornadas o el texto informativo, introducir los resultados y eliminar esta competición.

A continuación solo se detalla el apartado de jornadas, la introducción de resultados y el seguimiento del torneo ya que el resto de secciones (3.3.2.3, 3.3.2.4, 3.3.2.6, 3.3.2.7 y 3.3.2.8) son iguales a las de tipo liga.

3.4.3.1. Jornadas

En esta página no se mostrarán todas las jornadas como en la liga, ya que al ser un torneo eliminatorio, se debe conocer el resultado de los enfrentamientos para construir los emparejamientos de la siguiente. Además, se dividirán éstos según correspondan al torneo principal o al secundario (véase **Figura 3.22**). Esto es útil para llevar un seguimiento de cómo están yendo los enfrentamientos. También

la página muestra un mensaje para informarte de que en caso de que hubiese jugadores ficticios, éstos tendrían el nombre de *Ghost*.

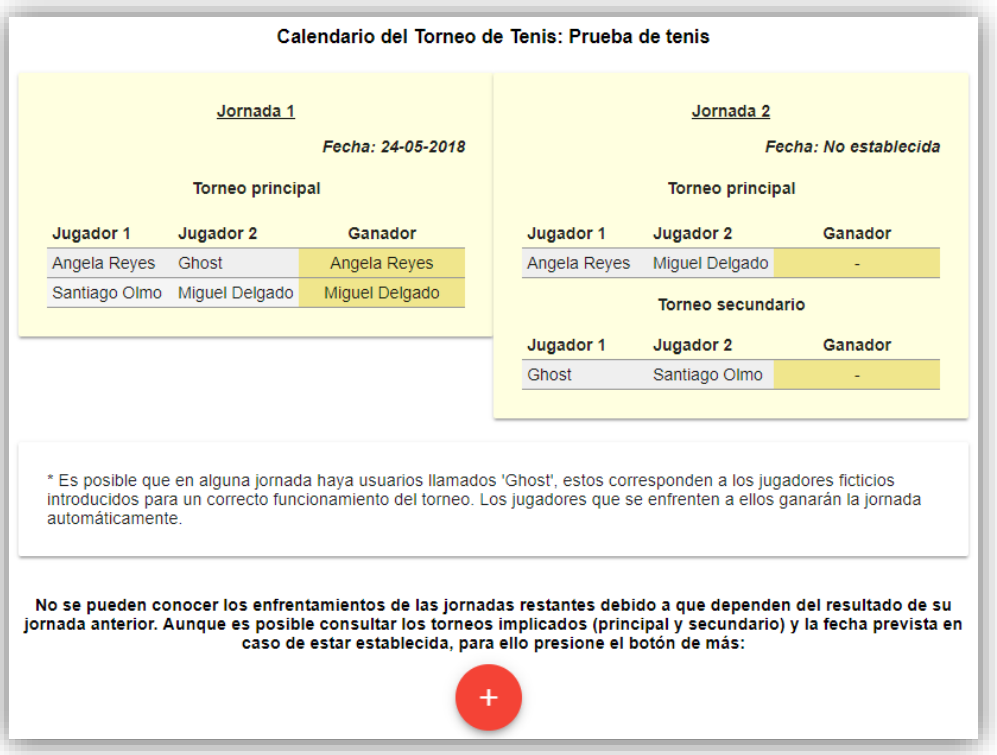

Figura 3.22. Sección 1 del calendario del torneo de tenis (4 participantes)

Sin embargo, aunque sea necesario conocer los resultados de las jornadas anteriores, existe la posibilidad de consultar las jornadas restantes para llevar un seguimiento de la fecha y de los torneos que participan en cierta jornada ya que no en todas se enfrentan ambos. En la **Figura 3.23** se observa un ejemplo.

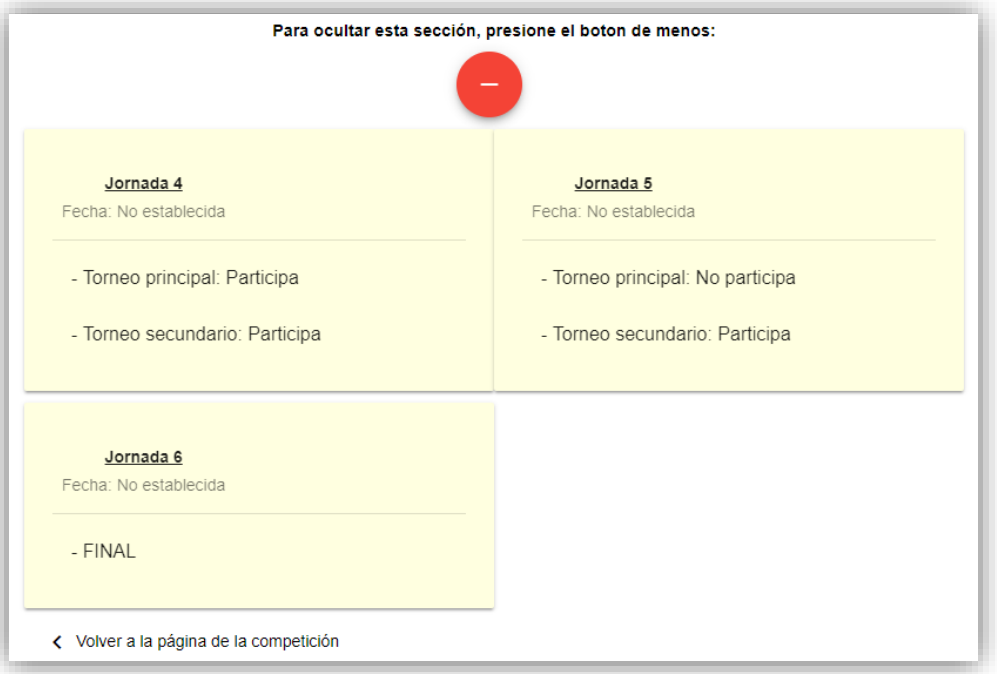

Figura 3.23. Sección 2 del calendario del torneo de tenis (8 participantes)

3.4.3.2. Seguimiento del torneo

Aunque al consultar las jornadas se pueda ver qué participantes siguen en el torneo principal y en el secundario, es más fácil verlo directamente desde esta página, que se irá actualizando conforme vayan teniendo lugar los enfrentamientos y vayan quedando eliminados los jugadores.

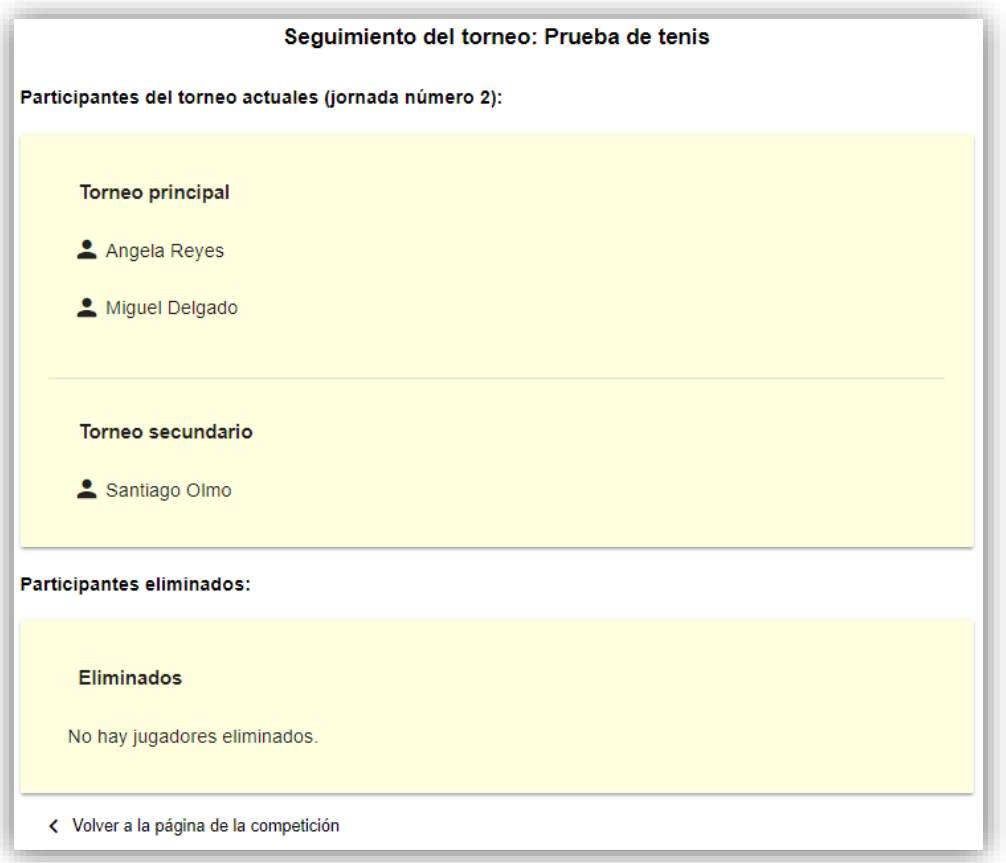

Figura 3.24. Seguimiento del torneo de tenis

Esta sección muestra una lista, de la jornada actual, con los alumnos que resisten en el torneo principal, otra con los que se mantienen en el secundario y otra con los estudiantes eliminados (véase **Figura 3.24**). De esta forma tanto alumnos como profesores pueden realizar un seguimiento del estado en el que se encuentran en la competición. En caso de haber jugadores ficticios añadidos, éstos no se muestran en esta sección debido a su irrelevancia y desinterés.

3.4.3.3. Introducción de resultados

La introducción de resultados en el torneo de tenis difiere con la de la liga en el método manual. Igualmente, se muestra la lista de duelos de alumnos o equipos que participan en la jornada actual pero existen varias diferencias. Estos enfrentamientos se dividen en torneo principal y/o secundario dependiendo al que pertenezcan, además, en el caso de que en alguno de estos participe algún jugador fantasma, la aplicación te lo hace saber y no te muestra dicho

enfrentamiento ya que es una victoria asegurada en esa jornada para el participante que compita contra este jugador ficticio.

La **Figura 3.25** es un ejemplo de cómo se vería este apartado en la aplicación.

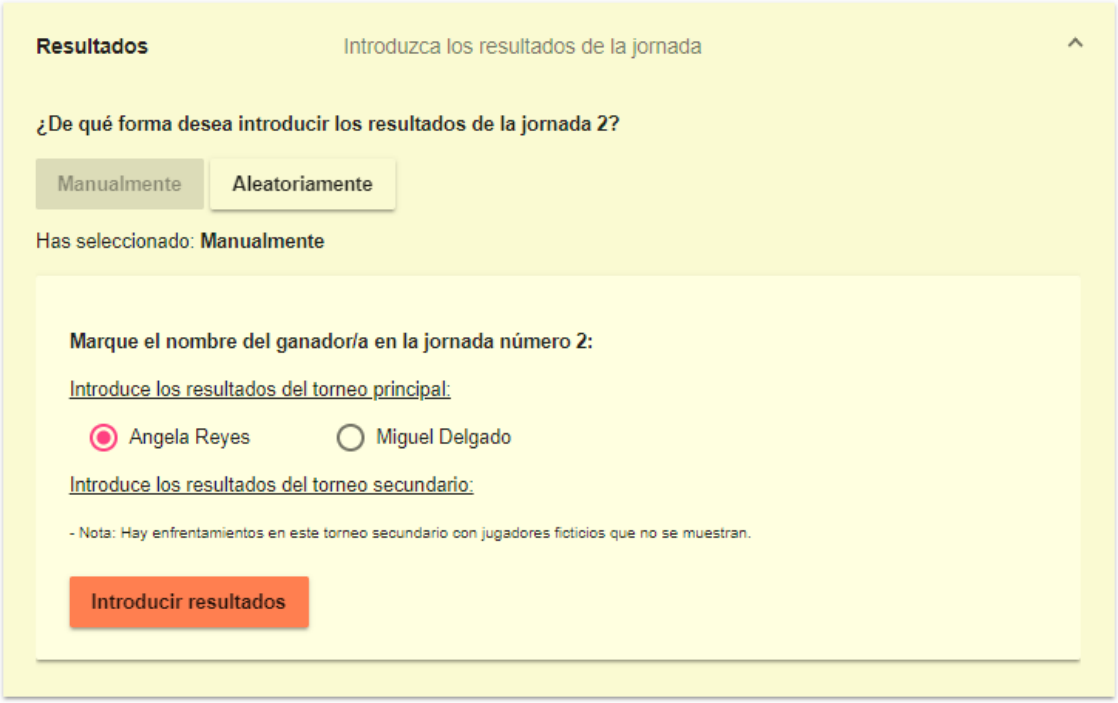

Figura 3.25. Introducción de resultados de forma manual (torneo de tenis)

Una vez que el profesor introduce todos los resultados de una jornada, o selecciona el modo aleatorio, automáticamente se crean los emparejamientos de la siguiente, a no ser que el torneo haya finalizado.

CAPÍTULO 4. MÓDULOS ADICIONALES: CREACIÓN DE EQUIPOS Y APLICACIÓN MÓVIL

Como se ha comentado en el capítulo anterior, las competiciones pueden ser individuales o por equipos. Para este último caso, es necesaria la existencia de una herramienta complementaria que cree los equipos.

Paralelamente a este proyecto, existe otro dedicado exclusivamente a esto pero debido a que aún no está terminado, ha sido necesaria la creación de este módulo provisional para poder llevar a cabo la realización de las competiciones por equipos. En este capítulo se explicarán las funcionalidades de esta herramienta adicional, además de las de la aplicación móvil.

4.1. Sección de creación de equipos

Para crear una competición por equipos, es necesario haber creado previamente los equipos. Para ello, se accede a la sección de creación de equipos a través de la barra de navegación (véase en la parte superior de la **Figura 4.1**). A ésta solo podrán acceder los profesores, sino mostrará un mensaje informativo.

4.1.1. Inicialización de los equipos

Una vez dentro de esta sección, lo primero que se ha de hacer es elegir el grupo del cual desea crear los equipos y darles nombre. Los grupos disponibles para seleccionar serán a los que pertenece el profesor.

Como se aprecia en la **Figura 4.1**, puedes añadir tantos equipos como quieras y eliminar alguno en concreto en caso de que hayas agregado más de la cuenta.

Existe la restricción de crear mínimo dos sino no se podrá continuar al paso siguiente, por tanto, al cargar esta página aparecerán dos equipos desde un principio y no existirá la posibilidad de eliminarlos.

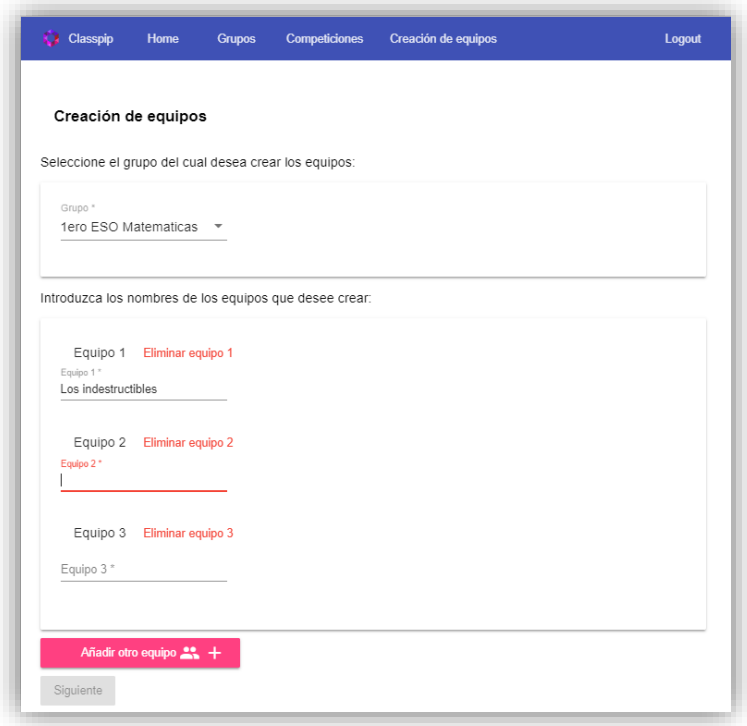

Figura 4.1. Sección de creación de equipos (paso 1)

4.1.2. Distribución de participantes

Una vez terminado el paso anterior, lo siguiente es repartir a los participantes del grupo elegido entre los equipos creados.

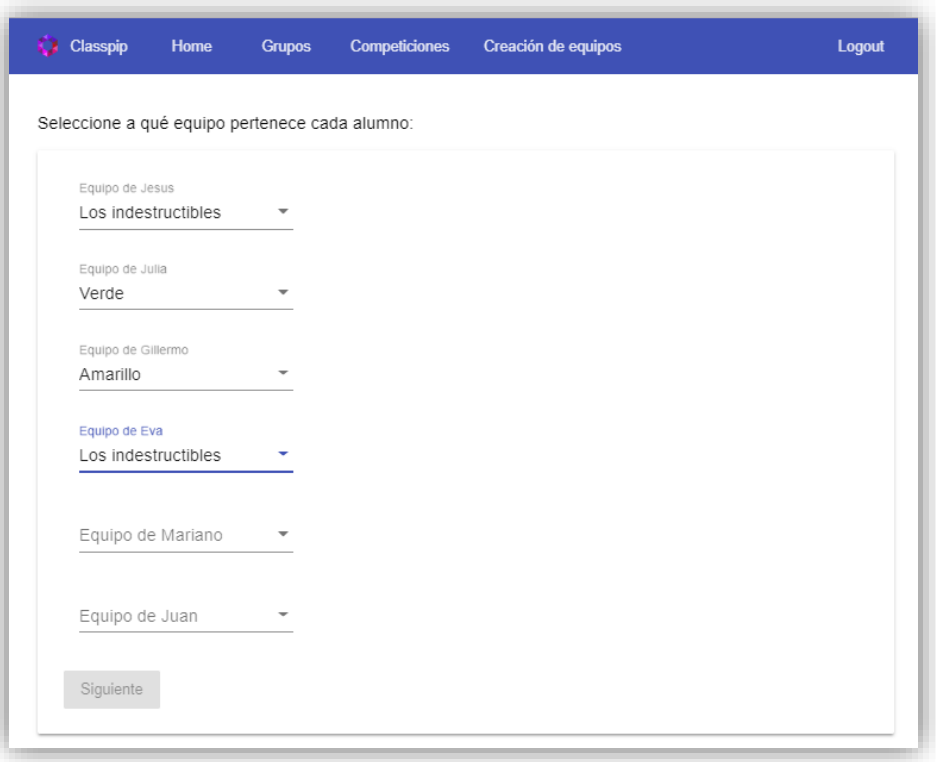

Figura 4.2. Sección de creación de equipos (paso 2)

Como se observa en la **Figura 4.2**, aparecerá una lista con todos los estudiantes pertenecientes al grupo escogido para poder determinar a qué equipo pertenece cada uno de ellos.

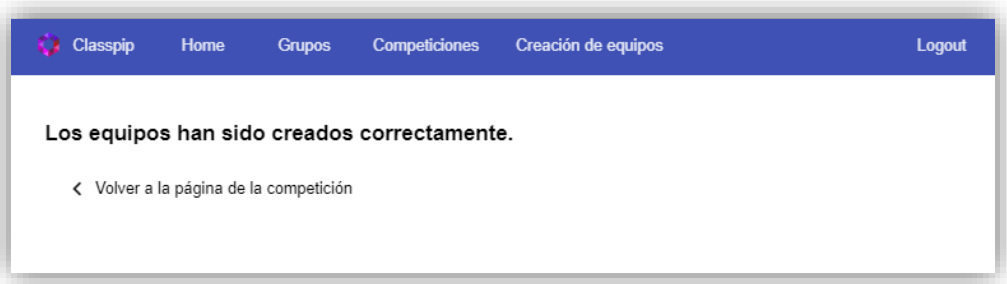

Figura 4.3. Sección de creación de equipos (terminado)

Completado este paso, ya estarían creados los equipos con sus alumnos correspondientes, por lo que aparecería el mensaje confirmándolo como en la **Figura 4.3**, desde donde ya podría redirigirse a la sección de competiciones.

En el **anexo 9.2** se encuentra la recopilación de las imágenes en el dashboard de esta sección en un tamaño mayor.

4.2. Aplicación móvil

Desde la aplicación móvil también se puede acceder a la sección de competiciones pero sólo para consultar información sobre éstas, no tiene tantas funcionalidades como el panel de administración. En el **anexo 9.3** están recopiladas todas las figuras de la aplicación móvil.

4.2.1. Funcionalidades

Desde el menú, se accede a la **sección de competiciones**, la cual se observa en la **Figura 4.4**, que mostrará una lista con las competiciones de las que forman parte los profesores o los alumnos. Una vez se seleccione una competición, la aplicación mostrará el **menú principal** de ésta, donde se encuentran todas las opciones posibles de consulta deseada. En la **Figura 4.5**, se muestra el menú para una liga por equipos. Para el torneo de tenis, en vez de **clasificación**, **Figura 4.6**, se puede consultar el **seguimiento del torneo**, **Figura 4.7**.

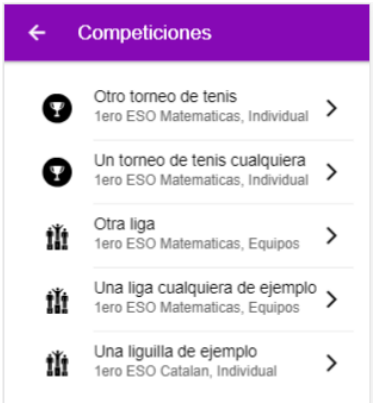

Figura 4.4. Sección de competiciones

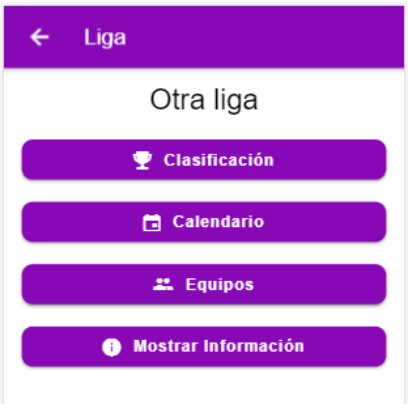

Figura 4.5. Menú principal de liga

Figura 4.6. Clasificación de la liga

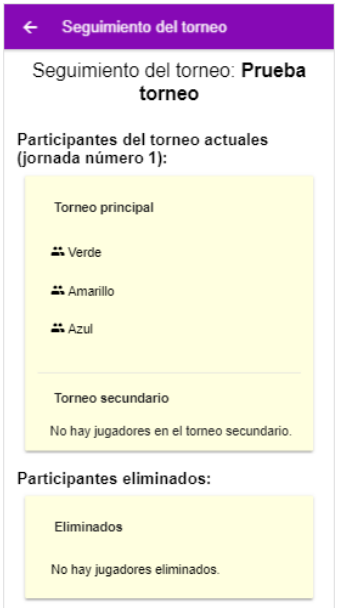

Figura 4.7. Seguimiento del torneo

En la **Figura 4.8** se observa el **calendario** tanto de la liga (a la izquierda) como el del torneo de tenis (derecha), que como se vió anteriormente, también se encuentra dividido en dos secciones.

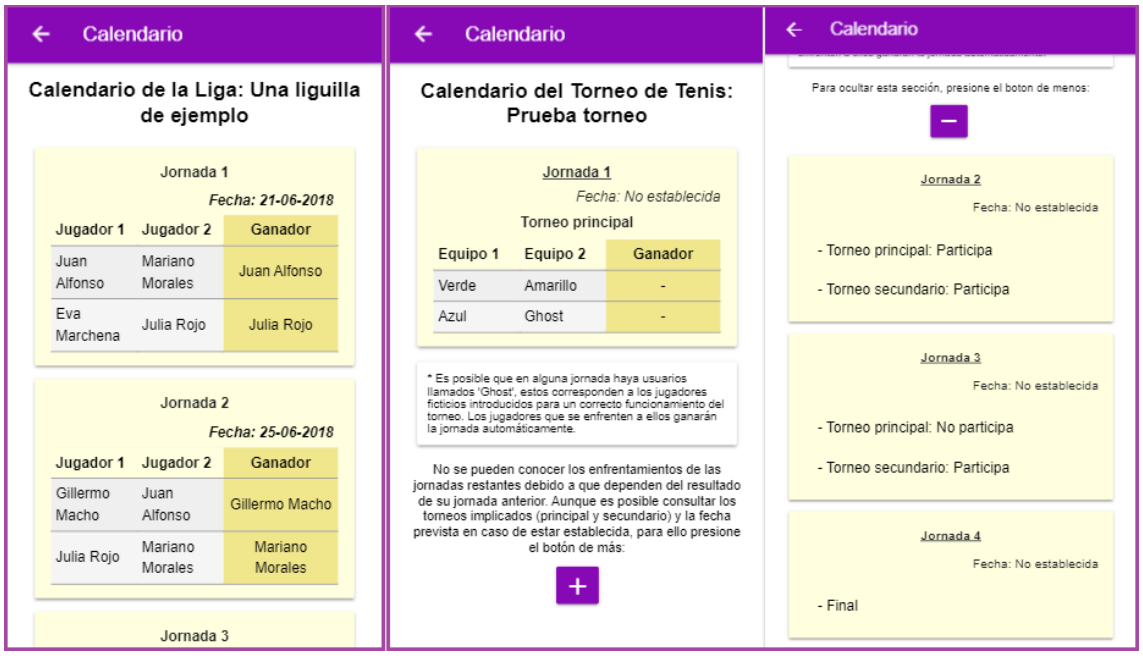

Figura 4.8.Calendario de la liga (izquierda) y del torneo de tenis (derecha)

Por último, la opción de **consultar los equipos** y la **información** acerca de la competición se muestra en la **Figura 4.10** y la **Figura 4.9** respectivamente.

Figura 4.10. Sección equipos

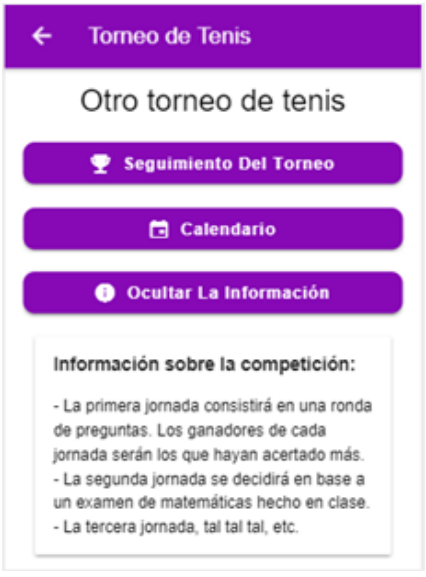

Figura 4.9. Información torneo

Las características de todas estas funcionalidades son exactamente iguales a las que tienen en el panel de administración y por tanto es redundante su explicación detallada.

CAPÍTULO 5. DESARROLLO DE LA APLICACIÓN

Con los objetivos claros sobre lo que se desea realizar y siendo conscientes de las bases desde las que se parte, es hora de empezar a implementar código.

Este capítulo se dividirá en cuatro partes, una por cada componente de la arquitectura software, ignorando la web pública, debido a que ésta no será modificada. Se explicará lo que se ha debido añadir en cada uno de estos componentes para cumplir con los objetivos propuestos en el **CAPÍTULO 3**.

5.1. Arquitectura orientada a servicios

Esta arquitectura orientada a servicios es una pieza clave sobre la que gira toda la aplicación ya que se encarga de tramitar todas las peticiones del usuario permitiendo la comunicación entre los diferentes componentes de software a través de una **interfaz de programación de aplicaciones (API)**. Gracias a esto, los datos pueden ser proporcionados a las diferentes plataformas del front-end (web y móvil) a la vez que éstas pueden enviar al servidor nuevos datos o modificaciones.

5.1.1. Interfaz de programación de aplicaciones

Con la API es posible realizar un servicio back-end capaz de funcionar en cualquier frontal, y por tanto, será utilizado por el cliente mediante el móvil y mediante el panel de administración por web. Para su creación, se utiliza un framework de Node.js de código abierto llamado **Loopback** [38] ya que permite desarrollar APIs de forma avanzada de una manera sencilla y conectarlas a fuentes de datos back-end. En la **Figura 5.1** puede verse su logo.

Figura 5.1. Logo de Loopback

Loopback ofrece una gran cantidad de utilidades pero solo se mencionarán las que han sido empleadas para la creación de este servidor. En los siguientes apartados podrán verse algunas de éstas.

5.1.2. Modelos

Los modelos son clases simples de JavaScript que representan la fuente de datos back-end. En el **apartado 2.1.2** se mostraban los principales modelos con los que contaba la aplicación pero con ellos no es suficiente, es necesario añadir otros nuevos para la creación del módulo de competiciones.

Construido encima de Express, Loopback permite crear un modelo y generar automáticamente una **API REST**, que puede ser llamada por cualquier cliente, con un conjunto de operaciones como crear, leer, actualizar y eliminar.

Al crear un nuevo modelo, se originan dos nuevos documentos: “nombremodelo.js” y “nombremodelo.json”. En el primero se crean los métodos remotos del modelo, hooks, etc. El segundo en cambio es el esqueleto del modelo donde se definen sus propiedades, relaciones, permisos, etc. La **Figura 5.2** es una parte del documento .json donde se establecen las propiedades del modelo Journey (jornada) que se verá a continuación.

Figura 5.2. Propiedades del modelo journey

Los nuevos modelos, junto con sus propiedades, que se han debido de incorporar son los siguientes:

- **Competición (competition)**: nombre, tipo (liga o torneo de tenis), modo (individual o por equipos), número de jornadas e información (las aclaraciones que el profesor establece sobre las bases de la competición).
- **Jornada (journey)**: número de jornada y fecha.
- **Enfrentamiento (match)**: jugador 1, jugador 2 (ambos serán la id de cada estudiante o de cada equipo que disputen ese enfrentamiento o un cero si se trata de un jugador ficticio) y ganador (corresponderá a la id del participante ganador, será 0 en caso de no haberse disputado ese duelo, será 1 si el resultado es empate o 2 si corresponde al descanso).
- **Equipo (team)**: nombre.

Además Loopback debido a la propiedad idInjection de los archivos .json, les agrega automáticamente una propiedad más a estos modelos: un identificador (id), que corresponde con la clave primaria en la base de datos en la que están conectada.

5.1.3. Relaciones

Estos modelos individuales pueden ser independientes los unos de los otros pero en muchas ocasiones tienen algún tipo de relación entre ellos. Es decir, pueden estar conectados entre sí.
Existen diferentes tipos de relaciones, las utilizadas en este proyecto han sido **BelongsTo**, **HasMany** y **HasAndBelongsToMany**:

- **BelongsTo**: relaciona un modelo con otro con una conexión 1:N, es decir, este modelo pertenecerá solamente a otro. Por ejemplo cada enfrentamiento pertenecerá a una sola jornada.
- **HasMany**: es el contrario a BelongsTo con una conexión 1:N, un modelo puede tener asociados varias instancias de otro modelo. Como ejemplo, una jornada tendrá asociados varios enfrentamientos.
- **HasAndBelongsToMany**: se trata de una conexión N:N. Un primer modelo puede relacionarse con varias instancias de un segundo modelo y viceversa. Un estudiante puede pertenecer a varias competiciones al igual que una competición tendrá más de un estudiante.

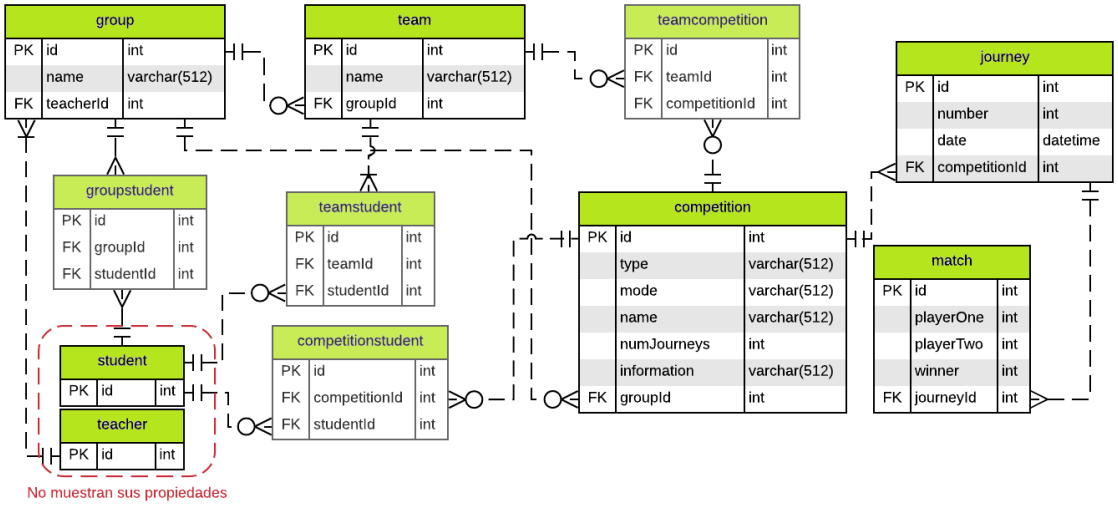

Figura 5.3. Diagrama entidad-relación

Observando la **Figura 5.3**[5] se puede deducir lo siguiente sobre las relaciones establecidas entre los modelos:

- **HasMany:**

  1. Una competición debe tener varias jornadas.
  2. Una jornada debe tener varios enfrentamientos.
  3. Un profesor puede pertenecer a un grupo o más.

[5] En esta figura se introducen los términos PK y FK y las tablas teamstudent, teamcompetition, groupstudent y competitionstudent, que se añaden para relacionar modelos en la base de datos. Se comentará con más detalle en el apartado 5.1.5.

4. Un profesor puede no haber creado ninguna competición o haber creado una o más (relacionado a través de su grupo).
5. Un profesor puede no haber creado ningún equipo o haber creado uno o más (relacionado a través de su grupo).

- **BelongsTo**:

1. Una jornada debe pertenecer únicamente a una competición.
2. Un enfrentamiento debe pertenecer únicamente a una jornada.
3. Un grupo debe pertenecer únicamente a un profesor.
4. Una competición debe haber sido creada y debe ser administrada únicamente por un profesor (relacionado a través de su grupo).
5. Un equipo ha sido creado por un profesor y es accesible únicamente por él (relacionado a través de su grupo).

- **HasAndBelongsToMany**:

1. Una competición debe de estar formada por más de un estudiante o no tener ningún estudiante (sólo cuando es modo de equipo, realmente siempre habrá estudiantes en la competición pero se relacionarán indirectamente a través de los equipos y no directamente como ocurriría en las competiciones individuales).
   Un estudiante puede participar en una o más competiciones o no participar en ninguna.
2. Una competición puede ser por equipos y que participen varios, o directamente no es por equipos y no tiene.
   Un equipo puede pertenecer a varias competiciones (que hayan sido creadas por el mismo profesor) o a ninguna en algún momento.
3. Un equipo debe de estar formado por uno o más de un estudiante.
   Un estudiante puede pertenecer a más de un equipo o no pertenecer a ninguno.
4. Un grupo debe estar formado por más de un estudiante.
   Un estudiante debería pertenecer a más de un grupo.

En el **anexo 9.4.1** pueden observarse las relaciones N:N que han debido de añadirse y en el **anexo 9.4.2** las relaciones 1:N. Para una visualización completa de todas las relaciones, en el **anexo 9.4.3** se encuentra el diagrama entidad-relación (ERD) de forma ampliada.

Todas estas relaciones son añadidas para el modelado y la gestión de la base de datos. De esta forma, Loopback añade un conjunto diferente de APIs y métodos a cada modelo dependiendo del tipo de relación para poder interactuar con ellos.

Es posible ver en la **Figura 5.4** un ejemplo de las líneas de código incorporadas en el archivo journey.json. El resto de modelos se encuentran en el **anexo 9.4.3**.

```
"relations": {
  "competition": {
    "type": "belongsTo",
    "model": "Competition",
    "foreignKey": "competitionId"
  },
  "matches": {
    "type": "hasMany",
    "model": "Match",
    "foreignKey": "journeyId"
  }
},
```

Figura 5.4. Relaciones del modelo journey.json

5.1.4. Permisos y seguridad

Una vez definidos los modelos y sus relaciones, se necesita añadir restricciones a éstos para controlar quien puede acceder a los datos o llamar a los servicios. Cada endpoint[6] o recurso de la API puede controlarse con el sistema de ACL a través de los archivos JSON de cada modelo. La **Figura 5.5** es un ejemplo del código ACL añadido a journey.json.

```
"acls": [
  {"accessType": "*", "principalType": "ROLE", "principalId": "$everyone", "permission": "DENY"},
  {"accessType": "READ", "principalType": "ROLE", "principalId": "$authenticated", "permission": "ALLOW"},
  {"accessType": "WRITE", "principalType": "ROLE", "principalId": "$authenticated", "permission": "ALLOW"}
],
```

Figura 5.5. ACLs de journey.json

A través de estas líneas de código estamos declarando lo siguiente:

1. Todos los métodos y propiedades del modelo de jornada están restringidas para todos.
2. Se permite la lectura de los datos de las jornadas a los usuarios autentificados.
3. Se permite la escritura sobre las jornadas a los usuarios autentificados.

Los demás modelos creados (competición, enfrentamiento y equipo) tienen los mismos permisos que el modelo de jornada. En el **anexo 9.4.4** se encuentran los archivos JSON de los modelos modificados completos (con propiedades, relaciones y ACL).

5.1.5. Base de datos

Como se comentó en el apartado 5.1.3, al crear relaciones entre modelos, se modela una base de datos para su posterior gestión. La arquitectura de ésta se muestra en la **Figura 5.6**, donde solo se incluyen los modelos que intervienen en el módulo de competiciones.

[6] Los endpoints son las URLs de un API o un backend que responden a una petición.

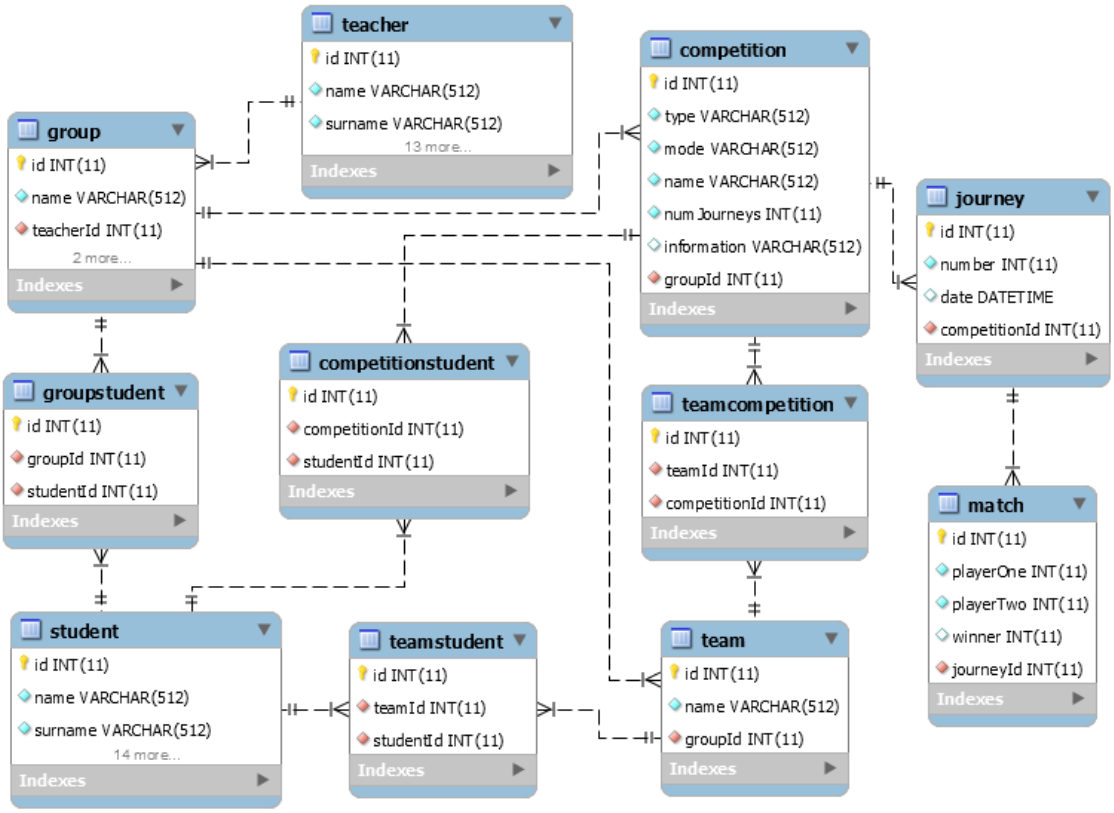

Figura 5.6. Arquitectura de la base de datos

Al generar una base de datos, se crean tablas de cada modelo que se conectan entre sí a través de sus id (primary key[7]):

- Si se trata de una relación 1:N (hasMany y BelongsTo), como por ejemplo entre la competición y la jornada, ésta última poseerá la id de la competición a la que pertenece (foreign key).
- En una relación N:N (hasAndBelongsToMany), como por ejemplo entre competición y estudiante, se crea una tabla intermediaria donde se relacionan ambas tablas mediante sus id.

Una vez se cree la base de datos, la API establecerá contacto con ésta para su gestión según las necesidades de los usuarios.

Para el desarrollo del código de la aplicación, es necesario configurar una base de datos local. Se ha reemplazado la configuración inicial con estas líneas de la **Figura 5.7** dentro del archivo datasources.json. De esta forma se controla mediante una base de datos local que todas las peticiones que se van realizando se ejecutan correctamente.

```
"db": {
  "host": "localhost",
  "port": 3306,
  "url": "",
  "database": "database-mdm",
  "password": "1994",
  "name": "db",
  "user": "mariader",
  "connector": "mysql"
},
```

Figura 5.7. Conector db

[7] Primary key (PK) se trata de la id propia de un modelo y foreign key (FK) consiste en la id que posee una tabla haciendo referencia a otra.

5.1.6. API Explorer

Loopback incorpora un cliente integrado llamado API Explorer con el que es posible testear las operaciones REST API que se van desarrollando. Gracias a esta herramienta se pueden obtener, agregar, modificar o eliminar datos y verificar la conexión con la base de datos.

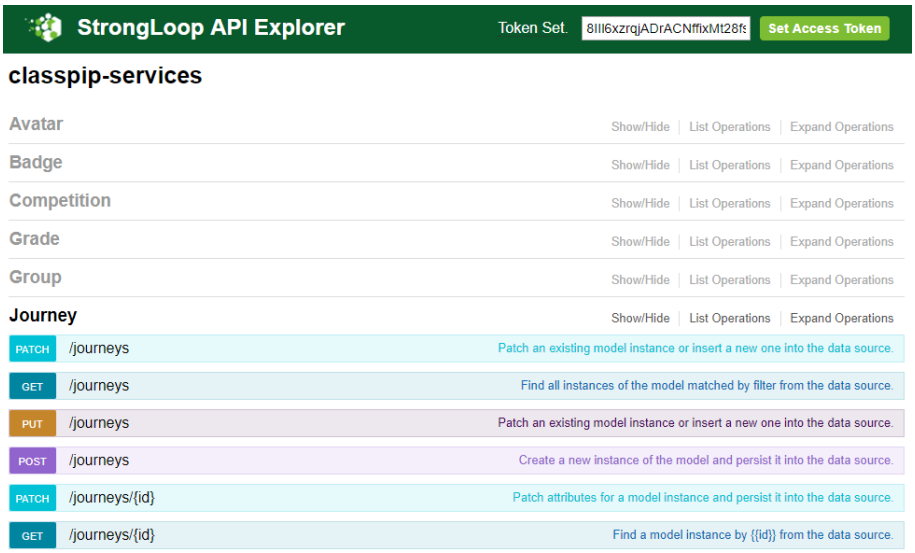

Figura 5.8. Interfaz del Strongloop API Explorer

Como se observa en la **Figura 5.8**, consiste en una interfaz donde se encuentran los endpoints de cada modelo, los cuales corresponden a una función específica como puede ser consultar los datos de una jornada en concreto o modificar la información de una competición. Cada modelo tendrá un conjunto de métodos diferentes dependiendo de las relaciones establecidas con otros modelos.

5.1.7. Boot Script

Por último, es necesario añadir más **estudiantes** al inicializar esta aplicación (bootstrapping) para así poder comprobar el funcionamiento de la aplicación en la realización de competiciones con un número mayor de alumnos. Además, también es necesario crear algunos **equipos** por defecto para que su id comience por un número superior al uno, en este caso por 10, y así garantizar que funciona correctamente el código desarrollado en la aplicación (en el **apartado 5.2** se explicará por qué ha sido necesario incluir esto).

Para ello, se han agregado algunas líneas de código en el archivo '06-create-students.js' del directorio /server/boot, para que LoopBack bootstrapper[8] ejecute esta script de arranque y adjunte estos modelos a la base de datos. En el **anexo 9.4.5** se encuentra el código que crea los equipos junto con sus alumnos.

[8] Loopback bootstrapper (programa de arranque) se encarga de efectuar la inicialización de la aplicación realizando varias tareas como pueden ser: configurar las fuentes de datos, definir modelos personalizados, configurar modelos y adjuntarlos a la base de datos, etc.

5.2. Panel de administración

Es a través del panel de administración desde donde se realizan todas las funciones de gestión de las competiciones existentes en la aplicación y por tanto, es en este donde se ha debido de incorporar más código.

El estilo de arquitectura de software seguido consiste en el **Modelo-Vista-Controlador** (**MVC**) [39], que separa los datos de la aplicación (modelo), la interfaz de usuario (vista) y la lógica de control (controlador).

Para el desarrollo de este panel de administración, principalmente se han debido de añadir **modelos**, que corresponden a las clases de las instancias implicadas en la aplicación, **componentes**, los cuales contienen la lógica para interactuar con el cliente y forman las vistas de la interfaz del usuario, y **servicios**, que proporcionan diferentes funcionalidades a los componentes donde se inyecten.

Tanto los componentes como los servicios deben ser declarados por el módulo de angular, el cual ordena la aplicación en bloques. Este es el encargado de declarar los componentes (entre otros), proveer de servicios a éstos e incluso de exportar otros módulos, tanto de angular, internos o librerías externas.

5.2.1. Modelos

Estos modelos representan la información que el panel de administración maneja, ya que en ellos se almacena la información obtenida de la base de datos además de poder crear nuevos objetos. Se tratan de clases desarrolladas en TypeScript que establecen las propiedades que tendrán las instancias y contienen el constructor del objecto y los métodos para obtenerlo y modificarlo.

Algunas clases incluso poseen más propiedades de las que definen al objeto en la base de datos. Un ejemplo es el caso de Match (**Figura 5.9**), en el cual además de poseer las id de ambos jugadores y del ganador, podrá guardarse el nombre exacto de cada uno de ellos en sus propiedades, hecho que resulta muy útil para trabajar con ellos en la aplicación. Todos los modelos creados se encuentran recopilados en el **anexo 9.5**.

```
export class Match {

  private _id: string;
  private _playerOne: number;
  private _playerTwo: number;
  private _winner: number;
  private _journeyId: number;
  private _namePlayerOne: string;
  private _namePlayerTwo: string;
  private _result: string;
```

Figura 5.9. Propiedades de la clase Match

5.2.2. Componentes

Los componentes definen las vistas con las que el cliente interactuará en la aplicación gracias a tres tipos de archivo. Primeramente, consta de un **controlador**, que se trata de una clase escrita en lenguaje TypeScript y que contiene los datos y la lógica de la aplicación, ésta se encuentra asociada a una plantilla **HTML,** que diseña la arquitectura y da forma a la interfaz combinando html común con material propio de Angular. Además, junto con su archivo **scss** correspondiente, se le dota de estilo como puede ser el color, la localización de cada bloque que conforma la arquitectura de la plantilla, etc.

Existe un cuarto tipo de archivo en la construcción de un componente. A medida que se desarrolla el software, es posible testear el código gracias a una herramienta llamada **Karma** [40], que se encarga de brindar un entorno donde se ejecutan los test unitarios construidos sobre los archivos **.spec.ts** y se obtiene un feedback con el resultado de las pruebas.

En la realización de este proyecto se han creado 12 componentes distintos que implican diferentes secciones en la aplicación. Para ir de unas vistas a otras, se utiliza el servicio de **Router** que proporciona Angular, ayudando a definir rutas de navegación entre éstas, como por ejemplo las observadas en la **Figura 5.10**. Sólo se comentará y se mostrará lo más relevante de los componentes creados debido a la gran cantidad de código que ha debido de incluirse, pero en el **anexo 9.7** se encuentran recopilados todos para su visualización completa.

```
{ path: 'competition/league/:id', component: LeagueComponent, canActivate: [AuthGuard]},
{ path: 'competition/league/:id/classification', component: ClassificationComponent, canActivate: [AuthGuard]},
{ path: 'competition/league/:id/journeys', component: JourneysLeagueComponent, canActivate: [AuthGuard]},
{ path: 'competition/league/:id/teams', component: TeamsComponent, canActivate: [AuthGuard]},
```

Figura 5.10. Parte de las rutas de navegación definidas

5.2.2.1. Creación de una competición

La creación para cada tipo de competición consta de su propio componente, ya que la lógica empleada diverge en algunos aspectos. De esta forma también, al incorporar nuevos tipos de competición en un futuro, sería más fácil y el código no se acumularía y quedaría más limpio y organizado.

Se diferencian 4 partes muy claras en la creación de una competición que se corresponden con cada formulario empleado. Los dos primeros pasos son prácticamente iguales, primeramente se recogen todos los datos de inicialización (número de jornadas en caso de liga, grupo, modo, etc.), necesarios para que seguidamente, se muestren los participantes (estudiantes o equipos). Una vez seleccionados estos participantes, en caso de ser torneo de **tenis**, **se calcula el número de jornadas** que tendrá. Esto se logra buscando el exponente que consiga que la potencia de dos sea igual o mayor al número de participantes y multiplicándolo por dos, (véase **Figura 5.11**). Por ejemplo, si participaran 5 alumnos, el exponente que logra que la potencia de dos sea mayor a 5 es el 3, por tanto el número de jornadas sería el doble, es decir, 6 jornadas.

```
// Computing number of journeys and participants (Math.pow(2, exp))
for (let _p = 1; !this.exp; _p++) {
  if ( Math.pow(2, _p) >= this.selectedParticipants.length ) {
  this.exp = _p;
  this.newCompetition.numJourneys = this.exp * 2;
  }
}
```

Figura 5.11. Código cálculo de jornadas

Para finalizar con la recolecta de los datos, los dos siguientes formularios recogen las fechas de cada jornada y la información que se añada sobre la competición. Una vez completados estos pasos, se llama a la función encargada de ir realizando todas las peticiones para guardar toda la información en la base de datos. Solamente faltaría organizar y guardar los enfrentamientos, que es la parte en la que más diferencias existen entre ambos tipos.

Primeramente, se construye un *array* con todos los participantes agregando jugadores ficticios si hiciera falta. En el caso de la liga, se añade solamente un

fantasma si el número de jugadores es impar (correspondería al descanso) y si se tratara del torneo de tenis, se agregan los que hagan falta para completar el redondeo por exceso a la potencia de dos. La posición que ocupa cada jugador en este *array* es aleatoria y por tanto los emparejamientos en la primera jornada también lo serán. Estos seguirán la **mecánica de emparejar** el primero con el último, el segundo con el penúltimo y así sucesivamente hasta terminar.

En la creación del torneo de tenis solo se definen los emparejamientos de la primera jornada pero en la **liga**, se construyen los enfrentamientos para **todas las jornadas**. Siguiendo la mecánica mencionada, sólo es necesario cambiar la posición de los participantes en el *array*. Como se aprecia en la **Figura 5.12**, el que ocupa la posición número uno, permanece inmóvil durante todas las jornadas mientras que todos los demás se van desplazando a una posición anterior excepto el de la dos, que deberá pasar a ocupar el último puesto.

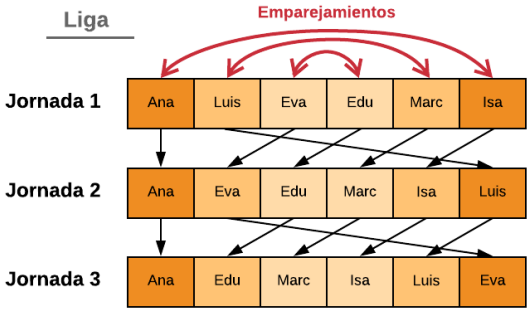

Figura 5.12. Emparejamientos (liga)

5.2.2.2. Resto de vistas

Otros de los componentes más complejos son los **menús principales** de cada tipo de competición. El menú no sólo te permite acceder al resto de vistas (clasificación, calendario, etc.) sino que es sobre este desde el cual se puede modificar el texto informativo o la fecha de las jornadas restantes e incluso introducir los resultados de los enfrentamientos. Respecto a esta última función, la aplicación busca la jornada que debe disputarse y solamente muestra los enfrentamientos que no contienen jugadores ficticios (debido a su trato especial); una vez introducidos los resultados de éstos, si se tratara de una competición de **tipo tenis**, automáticamente se **construirían los duelos** de la siguiente jornada. Para llevar a cabo esto, es esencial tener presente estas reglas:

1. En las **jornadas impares** sólo participará **un torneo**, el secundario, a excepción de la primera jornada en la cual es lógico que todos los duelos se encuentren en el principal.
2. En las **jornadas pares** participarán **ambos torneos** hasta llegar a la final, en la cual se enfrentan un jugador de cada torneo.

Gracias al cumplimiento de estas reglas es posible conocer en todas las jornadas **a qué torneo pertenece cada enfrentamiento**. Simplemente al construir los enfrentamientos de una jornada par, habrá que guardar en la base de datos antes los pertenecientes al principal y seguidamente los demás. De esta forma al acceder a estos enfrentamientos en un futuro, la primera mitad con **id** más baja corresponderán a los duelos del principal y el resto al secundario.

Conociendo estas reglas y la forma en la que los enfrentamientos se guardan, es posible desarrollar la mayoría de las funcionalidades del torneo de tenis que

posee la aplicación, como son mostrar los participantes en su torneo correspondiente, crear la sección del seguimiento del torneo y la construcción de los enfrentamientos de la siguiente jornada disputada. Se ha realizado una recopilación en el **anexo 9.6** del mecanismo de cada una de ellas con la finalidad de facilitar la comprensión del código implementado para su funcionamiento.

Respecto al sistema de puntuación de la **clasificación**, la aplicación analizará para todos los participantes la propiedad *winner* de sus correspondientes duelos. En caso de contener la **id** propia o la del otro jugador, se dará por ganado el encuentro sumando 3 puntos, o por perdido sin sumar nada respectivamente, un **1** sumaría un punto ya que se trata de un empate, un **2** corresponde al descanso de cierto participante en alguna jornada y por tanto ni suma puntos ni partidos jugados, y por último, un **0** significaría que esa jornada aún no se ha disputado y por tanto tampoco se realizaría ninguna operación. Como se comentó anteriormente en la sección 5.1.7, es necesario que la id de los equipos no comiencen a originarse a partir del número 1 ya que si no habría problemas al interpretar al ganador de los duelos. Se produciría cierta confusión entre si el 1 o el 2 se refieren al empate o al descanso o a la id de algún equipo.

Para terminar, sólo faltaría comentar un par de cosas. La primera es, que para ver las fechas en un formato adecuado no ha sido necesario construir ningún pipe ya que Angular consta de pipes predefinidos para transformaciones comunes, lo que ha permitido transformar los valores guardados en la base de datos para su visualización. La segunda y última, es relativa a todos los componentes en general, y es que para la visualización del texto de la aplicación se ha utilizado un traductor de por medio para que de esta forma, cuando en un futuro se agregue una herramienta multilenguaje, se pueda cambiar el idioma sin ningún problema, simplemente añadiendo el archivo .json a la carpeta de la internacionalización con la traducción al idioma deseado.

5.2.3. Servicios

Gracias a los servicios, se consigue separar el acceso a los datos de la presentación de los datos. Es cierto que dentro de un mismo componente podría incluirse la lógica necesaria para obtener los datos del servidor pero es preferible delegar estas tareas a los servicios.

La obtención de datos suele ir acompañada de un post-procesamiento de éstos y un manejo de errores entre otros, y por tanto, es conveniente su separación del componente para evitar sobrecargarlo y facilitar su comprensión. Además, al crear una clase de servicio, es posible compartir la lógica de esta entre los diferentes componentes que lo requieran y no ser exclusivo de una única vista.

En los servicios se han incluido todas las solicitudes que se realizan a la base de datos a través de peticiones **http**, como son la obtención, creación, modificación o eliminación de datos. En la **Figura 5.13**, se observa un ejemplo de un método público que debe ser llamado desde algún componente para realizar su función. En este caso, envía la id del grupo recibido como parámetro a un método privado del servicio que se encarga de realizar la petición http para obtener todas las

competiciones pertenecientes a ese grupo. Este primer método las obtiene mediante un **observable** (debido a su asincronía), les añade la clase y la materia correspondientes y las devuelve también a través de otro observable. Contiene manejo de errores por si surge cualquier problema en la obtención de los datos comunicárselo al componente que los ha solicitado.

.

```
public getMyCompetitionsByGroup(group: Group): Observable<Array<Competition>> {
  const ret: Array<Competition> = new Array<Competition>();
  return Observable.create(observer => {
    this.getCompetitionsByGroup(group.id).subscribe(competitions => {
      competitions.forEach(competition => {
        competition.grade = group.grade;
        competition.matter = group.matter;
        ret.push(competition);
        if (ret.length === competitions.length) {
          observer.next(ret);
          observer.complete();
        }
      });
    }, error => observer.error(error));
  });
}
/**
* This method returns the list of competitions by group
* @return {Array<Competition>} returns the list of competitions
*/
private getCompetitionsByGroup(groupId: string): Observable<Array<Competition>> {
  const options: RequestOptions = new RequestOptions({
    headers: this.utilsService.setAuthorizationHeader(new Headers(), this.utilsService.currentUser.id)
  });
  const url: string = AppConfig.GROUP_URL + '/' + groupId + AppConfig.COMPETITIONS_URL;
  return this.http.get(url, options)
    .map((response: Response, index: number) => Competition.toObjectArray(response.json()))
    .catch((error: Response) => this.utilsService.handleAPIError(error));
}
```

Figura 5.13. Parte de código del servicio de competiciones

5.3. Aplicación móvil

Gracias a que Ionic se basa en Angular, al estilo MVC seguido y a que las funciones de consulta que se realizarán mediante la aplicación móvil son las mismas que las que se realizan desde el dashboard, se ha podido reutilizar casi todo el código de los controladores y de los modelos cambiando pequeñas partes para adaptarlo. Para la vista, se han utilizado elementos de Ionic, que junto con el lenguaje HTML, consiguen transmitir una apariencia de aplicación nativa.

La **navegación** con el móvil es la diferencia más notable con respecto al dashboard, ya que no enruta hacia diferentes URLs que corresponden a determinados componentes. En este caso, la navegación se comporta como una pila simple en la que se colocan páginas en la parte superior para avanzar hacia delante en la aplicación, y para moverse hacia atrás, existe un botón de retroceso que se encarga de ir eliminando las páginas, también comenzando por las superiores. Además, incluso es posible el envío de objetos hacia la página a la que se desea navegar de una forma muy sencilla.

```
private goToCompetition(competition) {
  competition.type === 'Liga' ?
  this.navCtrl.push(LeaguePage, {competition: competition}) :
  this.navCtrl.push(TennisPage, {competition: competition});
}
```

Figura 5.14. Código de navegación

Como se observa en la **Figura 5.14**, sólo es necesario indicar la página donde se quiere navegar y el objeto a enviar. En este ejemplo, la aplicación navegará hacia la página indicada y le enviará como parámetro el objeto de la competición que se haya seleccionado. La página que añadirá a la pila será la correspondiente al menú de la liga o del torneo de tenis, dependiendo del tipo de competición a la que se refiera.

CAPÍTULO 6. HERRAMIENTAS ÚTILES EN EL PROCESO DE DESARROLLO DE SOFTWARE

Además de todas las herramientas descritas en el capítulo 2.2, hay más que se han utilizado como ayuda en este proceso de desarrollo del software. En este capítulo se comentarán cada una de estas.

6.1. GitKraken

Para gestionar fácilmente los repositorios de Git, se ha utilizado **GitKraken** [41] ya que se trata de una interfaz gráfica desde la cual podemos ejecutar todas las peticiones de Git como *pull*, *push*, crear *features*, *releases, etc.* Es posible desarrollar el código (creando diversas ramas, realizando commits, etc.) gestionándolo de forma local y una vez se desee subir a GitHub, solo habría que ejecutar un push y quedaría almacenado en la nube. La **Figura 6.1** muestra el aspecto que presenta esta aplicación.

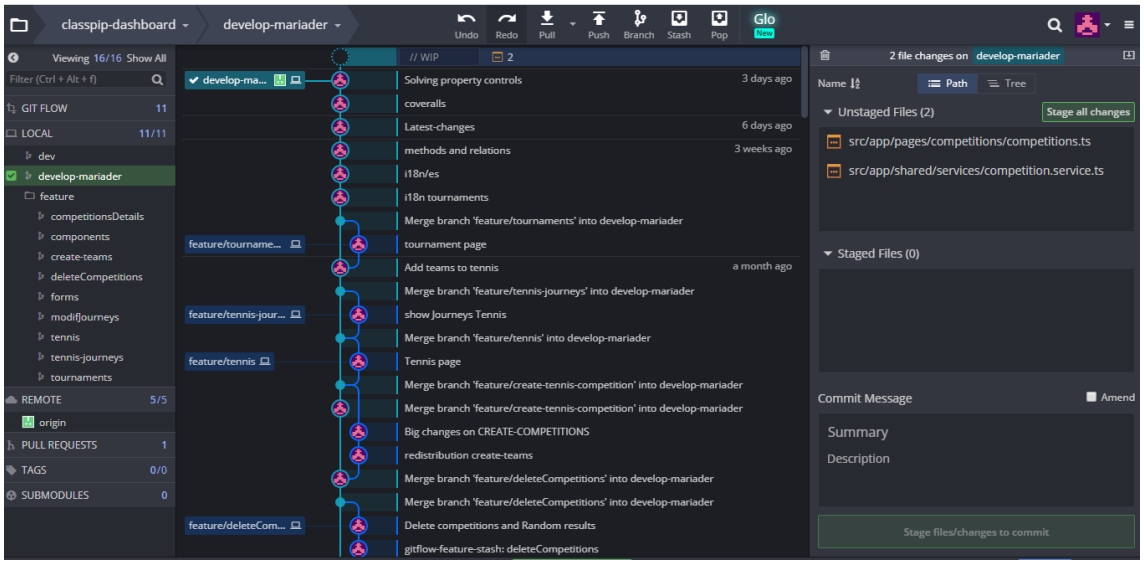

Figura 6.1. Interfaz gráfica de Git (GitKraken)

6.2. Insomnia

Al igual que el API Explorer, **Insomnia** [42] se trata de un cliente que te permite realizar todas las peticiones HTTP posibles del API REST. Esta herramienta me ha facilitado personalmente respecto al API Explorer, la comprensión de cómo están configuradas las peticiones HTTP ya que permite especificar una URL, los encabezados, la autorización y adjuntar la información que se desea enviar en su formato adecuado. De esta forma, es posible comprender las diferentes partes de las que se compone una petición HTTP y qué es lo que debe enviarse en cada una de ellas. Además, se obtienen todos los detalles en las respuestas y es posible ver qué se recibe exactamente y en qué formato, **Figura 6.2**.

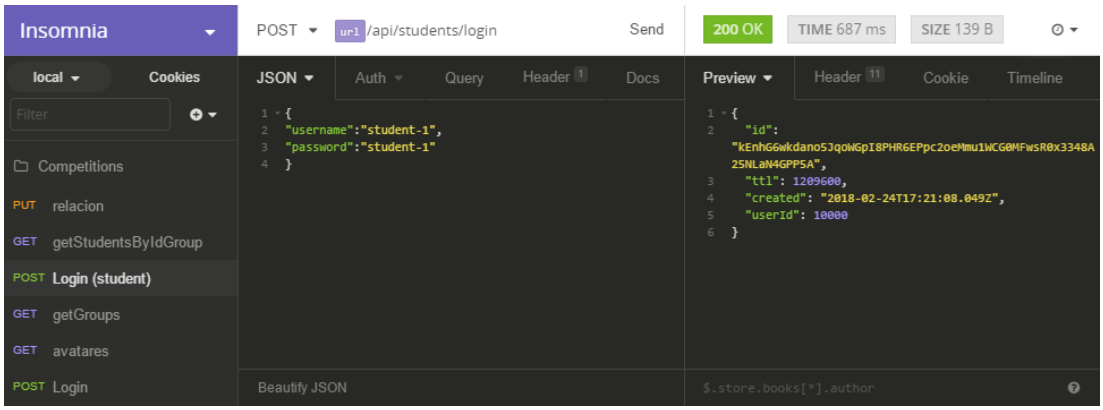

Figura 6.2. Interfaz de Insomnia

6.3. PhpMyAdmin

PhpMyAdmin [43] es una herramienta de software libre para gestionar bases de datos MySQL a través de una interfaz visual basada en la web (véase **Figura 6.3**), es una alternativa a **MySQL Workbench** [44], que se diferencia principalmente por ser una interfaz de escritorio. Contiene una larga lista de características, entre las cuales cabe destacar la posibilidad de explorar y alterar datos de las tablas que conforman la base de datos implicada. De esta forma, es posible comprobar el correcto funcionamiento de las peticiones que realiza la aplicación sobre la base de datos, como consultar si los datos se están generando y almacenando de la forma deseada, si se modifican correctamente o si se eliminan cuando se le ordene. También es muy útil a la hora de modificar un dato en concreto que la propia aplicación no permite realizarlo desde la misma, como podría ser cambiar el nombre de una competición creada.

Por último, ha facilitado la creación del diagrama ER, para representar los modelos de la base de datos junto con sus propiedades y relaciones.

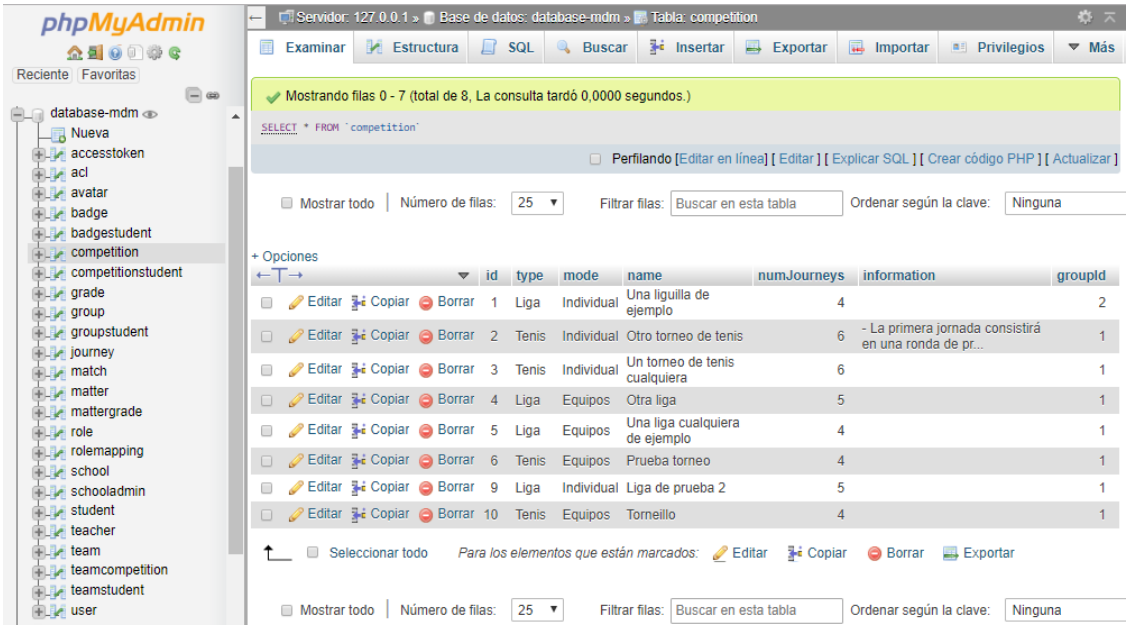

Figura 6.3. Interfaz de phpMyAdmin

CAPÍTULO 7. VALIDACIÓN DE LA APLICACIÓN

Este capítulo se encarga de validar la aplicación, verifica que todos los objetivos hayan sido cumplidos y proporciona una evaluación por parte de los usuarios.

7.1. Verificación de las funcionalidades

La realización de un **vídeo** demostrativo en el que se comprueban todas las funcionalidades de la aplicación, es de gran ayuda para **verificar** el cumplimiento de todos los objetivos propuestos. Por lo tanto, se ha elaborado un vídeo donde se explica paso a paso cómo funciona la aplicación a la vez que se demuestra. Para ello, se crean dos competiciones de cada tipo y se va interactuando con éstas. En el siguiente enlace: https://www.youtube.com/watch?v=CLrDNSkLqdU se encuentra el vídeo para su visualización.

Al realizarlo de esta forma, también actúa como videotutorial que **orienta al usuario** que se sirva de la aplicación a conocer el funcionamiento y las posibilidades de ésta.

Además, también serviría de utilidad para el **programador** que desee continuar desarrollando sobre esta herramienta, ya que detalla las bases de las que parte y cuál sería el correcto funcionamiento. Podría ir realizando los pasos que se siguen en el vídeo para comprobar que todo está correcto y que comenzará a programar sobre algo que funciona como debería.

A parte del vídeo, los **anexos 9.1**, **9.2** y **9.3,** contienen la recopilación de todo lo que la aplicación debe mostrar y realizar en sus diferentes casos, por lo que también es buena opción hacer uso de ella a modo de guía.

7.2. Evaluación de usuario

Por último, es fundamental conocer la opinión del cliente que hará uso de la aplicación, por tanto se ha realizado una encuesta (**anexo 9.10**) que permite obtener las primeras impresiones acerca de esta nueva herramienta incluida.

Es muy importante recibir este ***feedback*** para ser conscientes de las necesidades y posibles mejoras que se pudieran realizar, ya que, el principal objetivo, es crear una aplicación completa que consiga que el cliente quede satisfecho y se adapte a todas sus necesidades.

En esta encuesta, se adjunta el vídeo mencionado en el apartado anterior para que las personas que la realicen, conozcan el aspecto y las utilidades que se incluyen en la aplicación. Una vez sepan en qué consiste, ya podrán responder a las preguntas realizadas.

Son preguntas sobre el aspecto de la aplicación, la facilidad en su uso, preguntas de desarrollo para aportar nuevas ideas, etc. En la **Tabla 7.1** se recogen los datos obtenidos de las respuestas de 7 personas que contestaron. Las preguntas con respuestas de largo desarrollo no están incluidas.

Tabla 7.1. Respuestas sobre el cuestionario de Classpip

Pregunta sobre…	***Número de respuestas - Respuesta***
Aplicación en general	5 – Muy buena 2 - Buena
Estructura	7 – Muy buena
Facilidad de uso	6 – Muy fácil 1 - Fácil
Diseño gráfico	5 – Muy bueno 2 - Bueno
Tipo y tamaño de letra	6 – Muy bueno 1 - Bueno
Recomendabilidad	6 – Muy recomendable 1 - Recomendable
¿La utilizaría?	7 - Sí

Como puede observarse en la **Tabla 7.1**, respecto a la aplicación en general, su uso y su aspecto, las respuestas son muy positivas. De ahí que todos la utilizarían si tuviesen la ocasión además de recomendarla.

De entre todas las respuestas largas, destaca una opinión sobre la mejora en la introducción de los resultados. Considera importante incluir una herramienta que permita modificar los resultados de las jornadas puesto que es posible equivocarte a la hora de introducirlos.

También destacan otras como que en la creación de la competición, al avanzar al paso siguiente, sea posible retroceder al anterior para cambiar algún valor, o por ejemplo, que en la liga se puedan emparejar los duelos manualmente.

Por último, hay otra respuesta que sugiere que a la hora de elegir los participantes o equipos que van a participar en la competición, se añadiese la funcionalidad de seleccionar todos para ahorrar trabajo de ir seleccionando uno a uno.

CAPÍTULO 8. CONCLUSIONES Y TRABAJO FUTURO

8.1. Conclusiones técnicas

Antes de comenzar a desarrollar una aplicación, no sólo es necesario determinar el objetivo de ésta para definir las funciones necesarias y comenzar a escribir código. Sino que es fundamental analizar todas las herramientas existentes para el desarrollo de software y elegir las que más se adapten a nuestros objetivos.

Puesto que la intención inicial era la elaboración de una aplicación que incluyera unas funcionalidades para el escritorio y otras para el móvil, la elección de **Angular** como framework para el desarrollo del panel de administración web ha supuesto una gran cantidad de ventajas. Entre otras, destaca el haber podido reutilizar gran parte del código incluido en este dashboard en la aplicación móvil gracias a que ésta se ha creado con **Ionic**, que permite desarrollar aplicaciones móviles en base a tecnologías web (HTML5, CSS, JavaScript, etc.), pudiendo utilizarse además en cualquier sistema operativo con una apariencia nativa.

De esta forma, se facilita y reduce significativamente el coste y el tiempo de desarrollo debido a la reutilización de paquetes y código entre ámbos y su visualización en multitud de plataformas sin tener que realizar código exclusivo para cada una de ellas. Con lo cual, **es esencial hacer desde un principio una buena elección de las tecnologías** con las que se va a trabajar.

Es recomendable el **uso de un sistema de control de versiones**, como Git, que registra todos los cambios realizados sobre el código permitiendo conocer toda su evolución en el tiempo. De esta forma se consigue un control sobre el código, siendo posible regresar a un estado anterior del proyecto, comparar cambios, recuperar partes de la lógica eliminada e incluso detectar errores. Esto permite cierta organización y una gestión ágil y más fácil del proyecto.

Otro de los aspectos más importantes que debe mencionarse, es la creación de la aplicación con **software de código abierto** pensado para un desarrollo colaborativo. Esto permite seguir una metodología de programación de mejora continua en el que diferentes personas podrían desarrollar simultáneamente nueva lógica que añadir al código fuente original. Con esto, se consigue crear un entorno ágil, eficiente e innovador donde se aportan gran cantidad de nuevas ideas y soluciones para lograr una aplicación completa, funcional y de calidad, que no tenga nada que envidiar a las demás del sector, sino al contrario.

Se ha logrado cumplir con todos los objetivos definidos en este proyecto, desarrollando así un generador de competiciones para implementar en una aplicación destinada a la gamificación. Estos objetivos eran:

- Generador de dos tipos diferentes de competiciones
- Generador de equipos para insertar esta modalidad.
- Dotar de funcionalidades de gestión y consulta a las competiciones.

8.2. Conclusiones personales

Antes de comenzar con este proyecto, ya sabía que supondría todo un reto para mí debido a que se trata de un tema más próximo a las titulaciones de telecomunicaciones que a la mía de aeronavegación, pero me impulsé a hacerlo por dos motivos. Primero, me llamó mucho la atención el tema, poder participar en un proyecto así en el que hay tanta gente implicada para llevarlo a cabo me interesó mucho, y el otro, es que lo que he aprendido sobre programación durante la carrera, me ha encantado, por lo que me motiva a querer seguir aprendiendo. Además, es una oportunidad para demostrar la capacidad de aprendizaje y de adaptación a nuevos desafíos, ya que es algo a lo que todo ingeniero debe de estar preparado.

La primera y mayor dificultad encontrada fue al comienzo. Tener que entender el código desde el que vas a comenzar a desarrollar, cómo están relacionados todos los componentes que forman la arquitectura de la aplicación (servidor, dashboard, etc.) y todas las tecnologías implicadas en ella fue una tarea muy dura, sobretodo porque implica mucho tiempo dedicado al aprendizaje y no tanto en avanzar con tus objetivos.

Durante el desarrollo, he ido encontrándome con algunas dificultades, aunque quizá lo más destacable como para comentar, fue entender cómo tenían que construirse las peticiones http y adaptarme a trabajar con su asincronía, aprendiendo el uso y mecanismo de los observables.

El haber ido superando dificultades durante el desarrollo y haber acabado con estos resultados tan satisfactorios y, sobretodo, con tantos conocimientos nuevos adquiridos sobre tecnologías tan actuales, es realmente gratificante y hace que me sienta muy orgullosa y feliz.

8.3. Líneas abiertas

La implementación de este generador de competiciones creado, además de dotar a la aplicación de una nueva herramienta que aporta diversión y motivación extra en los alumnos, consigue **abrir muchas líneas de trabajo** debido a su posible integración con otros módulos o a la inclusión de mejoras internas.

Un ejemplo, sería su integración con el **módulo de preguntas**. Podría incluirse una herramienta en la competición que permita que los ganadores de alguna jornada se decidiesen en base a quiénes hayan acertado el mayor número de preguntas. Los resultados de esa jornada podrían introducirse automáticamente una vez se conozcan.

Además, también sería perfectamente integrable con el **módulo de puntos y** el **de colecciones de cartas**. Los ganadores de las competiciones, o de los duelos de cada jornada, podrían ser recompensados con puntos o cartas que se entregasen automáticamente al conocerse los resultados.

Otra integración que debe hacerse, es con el módulo dedicado a la **creación de equipos**, ya que el desarrollado en este proyecto es provisional debido a que los equipos deben de poseer más características y funcionalidades para poder extrapolarse a otros ámbitos y no solo a las competiciones.

Como **mejoras internas** en la sección de competiciones, pueden realizarse algunos cambios e introducir nuevas utilidades.

Una herramienta fundamental que debe mejorarse, es la introducción de resultados. Puesto que existe la posibilidad de equivocarse al introducir alguno, es necesario que pueda realizarse una **modificación en los resultados**.

Otro aspecto interesante, reside en la información de la competición. En ésta se redacta fundamentalmente, qué criterios se seguirán para decidir a los ganadores de los duelos en cada una de las jornadas. Por tanto, podría mantenerse la información general de la competición como se encuentra actualmente, y a parte, agregar a **cada jornada su propia información** donde se aclaren las bases de sus enfrentamientos. De esta forma, al mostrar cada jornada con su correspondiente información, sería mucho más visual y fácil de encontrar, quedando aún más claro.

La **creación de las competiciones** consta de una serie de pasos que siguen una única dirección, lo que conlleva a no poder retroceder hacia el paso anterior en caso de querer modificar algún dato, por tanto, también sería interesante añadir esa **bidireccionalidad** por si se quisiera cambiar algún dato introducido sin tener que volver a empezar de nuevo.

Por último, es esencial la inclusión de **otros tipos de competición** (carrera ciclista, sistema suizo, etc.) para el crecimiento de esta sección dedicada a las competiciones. Cuantas más haya, más posibilidades hay y más probabilidad de que se adapte al gusto del usuario.

BIBLIOGRAFÍA

[1] G. Zichermann and C. Cunningham, *Gamification by design*. 2011.

[2] Wikipedia, “Edgar Dale.” [Online]. Available: https://es.wikipedia.org/wiki/Edgar_Dale. [Accessed: 15-May-2018].

[3] Centro de Comunicación y Pedagogía, “Juego serio: gamificación y aprendizaje.” [Online]. Available: http://www.centrocp.com/juego-serio-gamificacion-aprendizaje/. [Accessed: 15-May-2018].

[4] Felipe Quintanal Pérez, “Gamifcación y la Física–Química de Secundaria.” [Online]. Available: http://www.redalyc.org/html/2010/201048819002/. [Accessed: 08-Jun-2018].

[5] “El plan del héroe.” [Online]. Available: http://theheroplan.com/es/metodo/. [Accessed: 15-May-2018].

[6] “ClassDojo.” [Online]. Available: https://www.classdojo.com/es-es/. [Accessed: 15-May-2018].

[7] “Ribbon Hero 2.” [Online]. Available: https://www.microsoft.com/en-us/microsoft-365/blog/2011/04/26/ribbon-hero-2-how-to-play-the-game-video/. [Accessed: 15-May-2018].

[8] “Kahoot.” [Online]. Available: https://kahoot.com/. [Accessed: 15-May-2018].

[9] “World Peace Game.” [Online]. Available: https://worldpeacegame.org/. [Accessed: 15-May-2018].

[10] “Zombie-Based Learning.” [Online]. Available: http://zombiebased.com/. [Accessed: 15-May-2018].

[11] “Socrative.” [Online]. Available: https://www.socrative.com/. [Accessed: 15-May-2018].

[12] Wikipedia, “Framework.” [Online]. Available: https://es.wikipedia.org/wiki/Framework. [Accessed: 17-May-2018].

[13] Wikipedia, “Interfaz de Programación de Aplicaciones.” [Online]. Available: https://es.wikipedia.org/wiki/Interfaz_de_programación_de_aplicaciones. [Accessed: 17-May-2018].

[14] Wikipedia, “Transferencia de Estado REpresentacional.” [Online]. Available: https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional. [Accessed: 17-May-2018].

[15] “StrongLoop.” [Online]. Available: https://strongloop.com/. [Accessed: 17-May-2018].

[16] “JavaScript.” [Online]. Available: https://www.javascript.com. [Accessed: 17-May-2018].

[17] “Express.” [Online]. Available: http://expressjs.com/es/. [Accessed: 17-May-2018].

[18] “Nodejs.” [Online]. Available: https://nodejs.org/en/. [Accessed: 17-May-2018].

[19] “MySQL.” [Online]. Available: https://www.mysql.com/. [Accessed: 17-May-2018].

[20] “Angular.” [Online]. Available: https://angular.io/. [Accessed: 17-May-2018].

[21] Wikipedia, “TypeScript.” [Online]. Available:

https://es.wikipedia.org/wiki/TypeScript. [Accessed: 17-May-2018].
[22] “Bootstrap.” [Online]. Available: https://getbootstrap.com/. [Accessed: 17-May-2018].
[23] “Angular Material.” [Online]. Available: https://material.angular.io/. [Accessed: 17-May-2018].
[24] “Ionic.” [Online]. Available: https://ionicframework.com/docs/. [Accessed: 03-Jun-2018].
[25] “Apache Cordova.” [Online]. Available: https://cordova.apache.org/. [Accessed: 03-Jun-2018].
[26] “Sass.” [Online]. Available: https://sass-lang.com/. [Accessed: 03-Jun-2018].
[27] “GitHub.” [Online]. Available: https://github.com/. [Accessed: 19-May-2018].
[28] “ZenHub.” [Online]. Available: https://www.zenhub.com/. [Accessed: 19-May-2018].
[29] Wikipedia, “Integración continua.” [Online]. Available: https://es.wikipedia.org/wiki/Integración_continua. [Accessed: 19-May-2018].
[30] “Coveralls.” [Online]. Available: https://coveralls.io/. [Accessed: 19-May-2018].
[31] “Codacy.” [Online]. Available: https://www.codacy.com/. [Accessed: 19-May-2018].
[32] “Travis CI.” [Online]. Available: https://travis-ci.org/. [Accessed: 19-May-2018].
[33] “Docker.” [Online]. Available: https://www.docker.com/. [Accessed: 20-May-2018].
[34] “Digital Ocean.” [Online]. Available: https://www.digitalocean.com/. [Accessed: 20-May-2018].
[35] “HockeyApp.” [Online]. Available: https://hockeyapp.net/. [Accessed: 20-May-2018].
[36] “Npm.” [Online]. Available: https://www.npmjs.com/. [Accessed: 20-May-2018].
[37] “New Relic.” [Online]. Available: https://newrelic.com/. [Accessed: 20-May-2018].
[38] “LoopBack.” [Online]. Available: https://loopback.io/. [Accessed: 17-May-2018].
[39] Wikipedia, “Modelo-vista-controlador.” [Online]. Available: https://es.wikipedia.org/wiki/Modelo–vista–controlador. [Accessed: 24-May-2018].
[40] “Karma.” [Online]. Available: http://karma-runner.github.io/2.0/index.html. [Accessed: 24-May-2018].
[41] “GitKraken.” [Online]. Available: https://www.gitkraken.com/. [Accessed: 20-May-2018].
[42] “Insomnia.” [Online]. Available: https://insomnia.rest/. [Accessed: 21-May-2018].
[43] “phpMyAdmin.” [Online]. Available: https://www.phpmyadmin.net/. [Accessed: 21-May-2018].
[44] “MySQL Workbench.” [Online]. Available: https://www.mysql.com/products/workbench/. [Accessed: 21-May-2018].

CAPÍTULO 9. ANEXOS

9.1. Imágenes de la sección de competiciones en el dashboard

9.1.1. Menú principal

En la **Figura 9.1** y la **Figura 9.2** se observa respectivamente el menú principal de las competiciones si accedes al panel de administración como profesor teniendo competiciones creadas y sin tener.

Figura 9.1. Menú principal para los profesores con competiciones

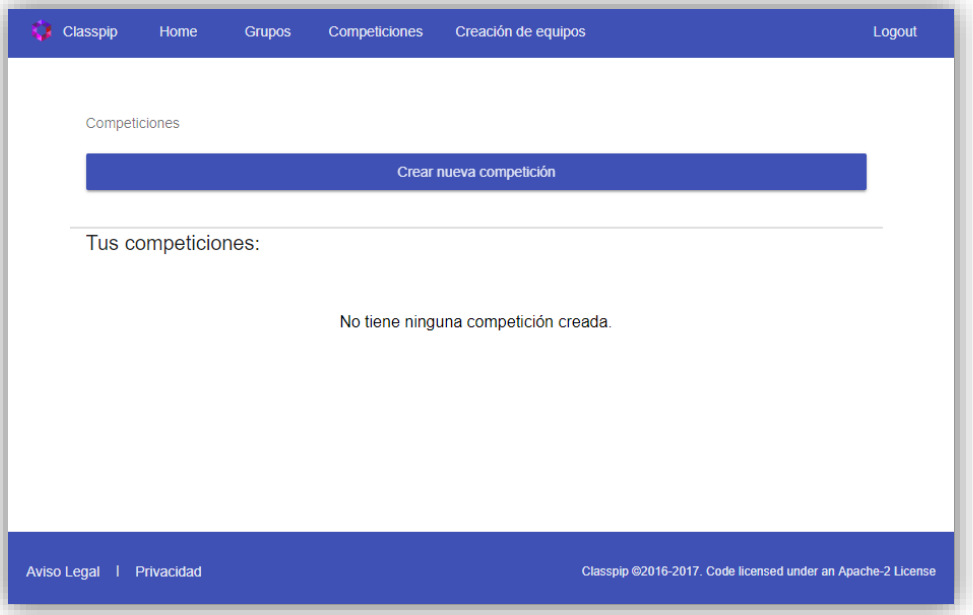

Figura 9.2. Menú principal para los profesores sin competiciones

En la **Figura 9.3** y **Figura 9.4** se observa respectivamente el menú principal de las competiciones si accedes en la aplicación como estudiante participando en competiciones y sin participar.

Figura 9.3. Menú principal para los estudiantes con competiciones

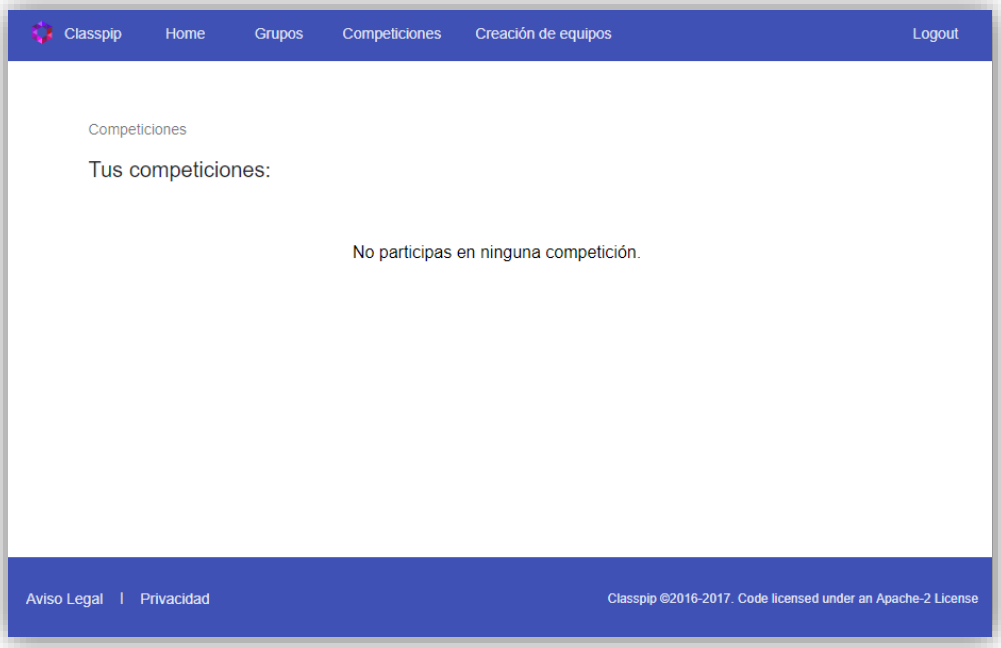

Figura 9.4. Menú principal para los estudiantes sin competiciones

9.1.2. Creación de una competición

En esta sección del anexo se recopilan todas las apariencias posibles en la creación de una competición.

9.1.2.1. Inicialización

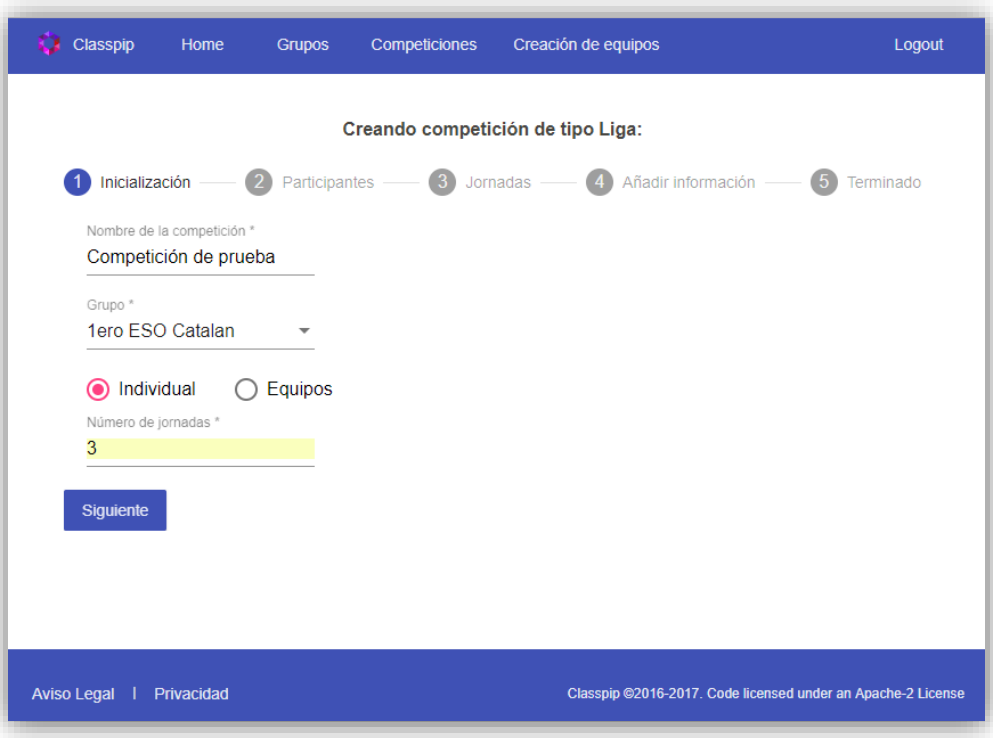

Figura 9.5. Creación competición de la liga (inicialización)

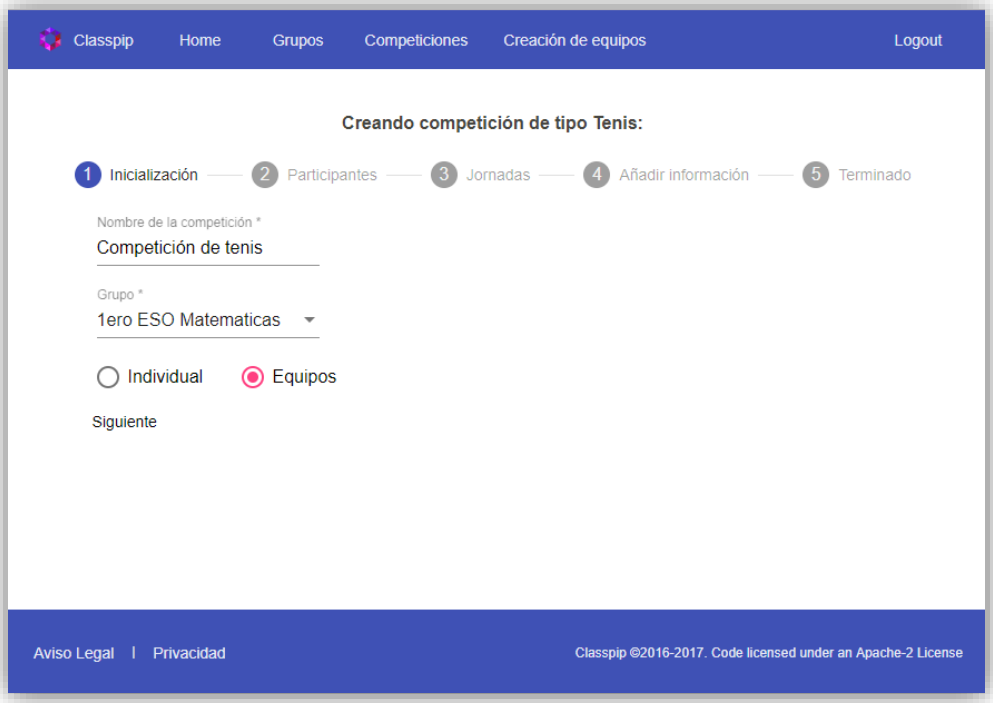

Figura 9.6. Creación competición del torneo de tenis (inicialización)

9.1.2.2. Participantes

En la **Figura 9.7**, al ser una competición individual, muestra el nombre de los estudiantes pertenecientes al grupo indicado en la inicialización. Si fuera por equipos, en lugar de los alumnos mostraría los nombres de los equipos.

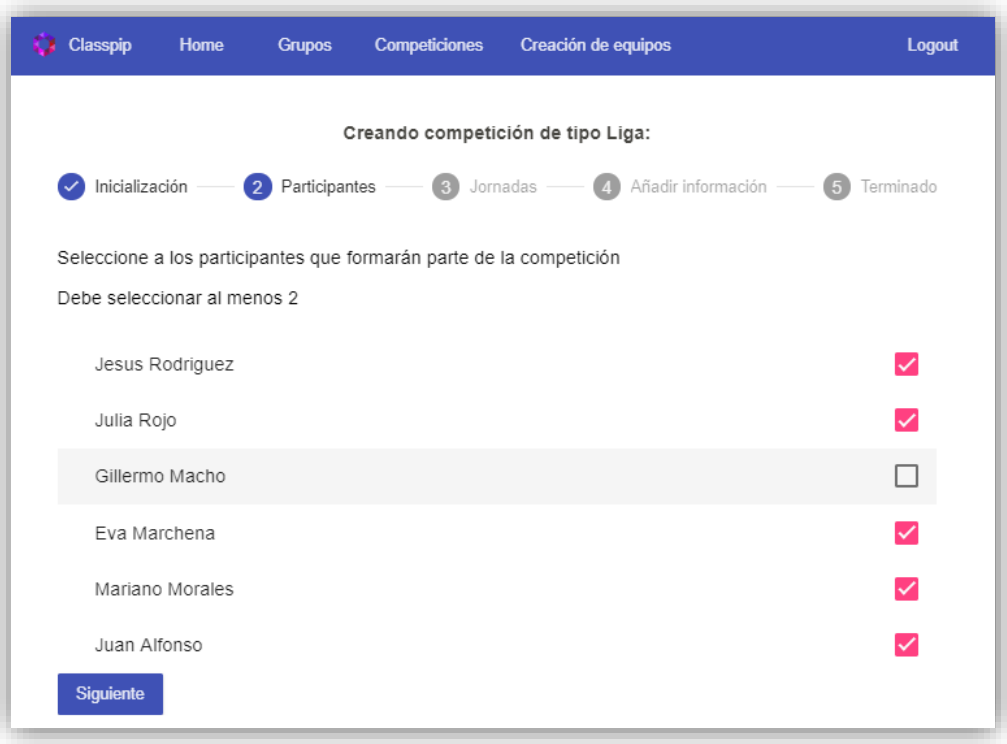

Figura 9.7. Creación competición (participantes)

9.1.2.3. Jornadas

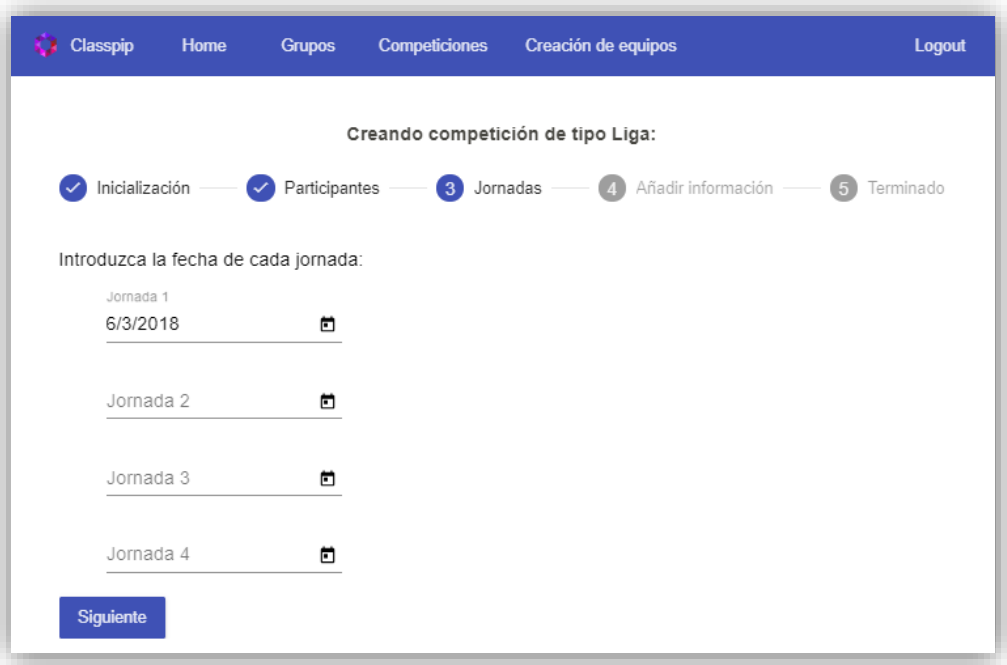

Figura 9.8. Creación competición (jornadas)

9.1.2.4. Añadir información

Figura 9.9. Creación competición (añadir información)

9.1.2.5. Terminado

Figura 9.10. Creación competición (jornadas)

9.1.3. Competición de tipo: Liga

En esta sección del anexo se recopilan todas las apariencias posibles de la liga, exceptuando las que comparte con el torneo de tenis.

9.1.3.1. Menú principal

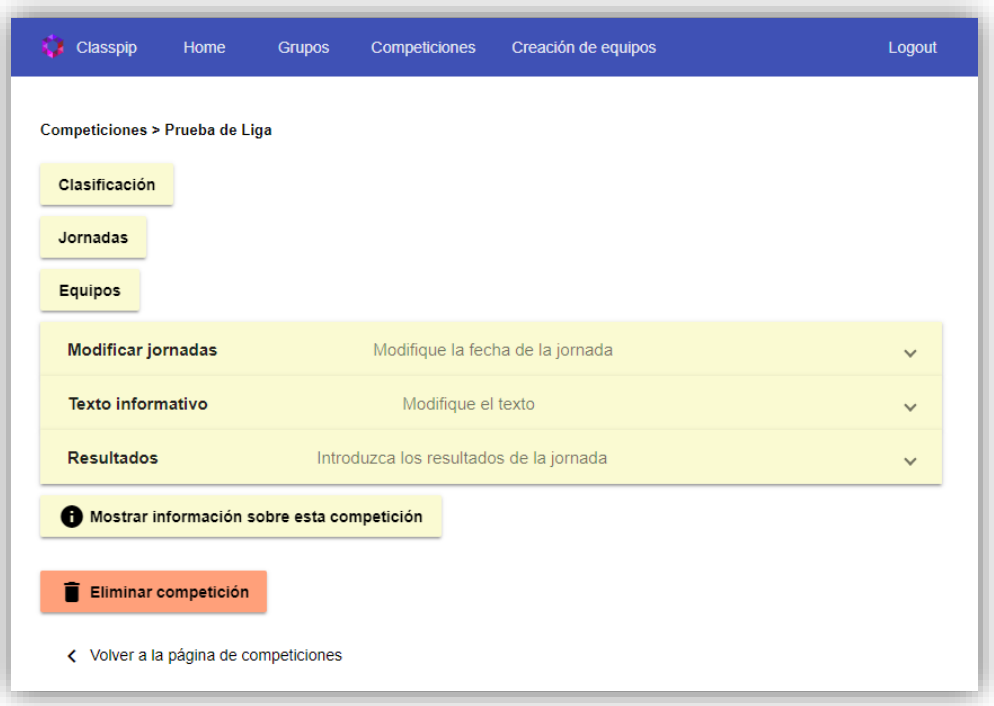

Figura 9.11. Menú principal de la liga modo equipos (profesores)

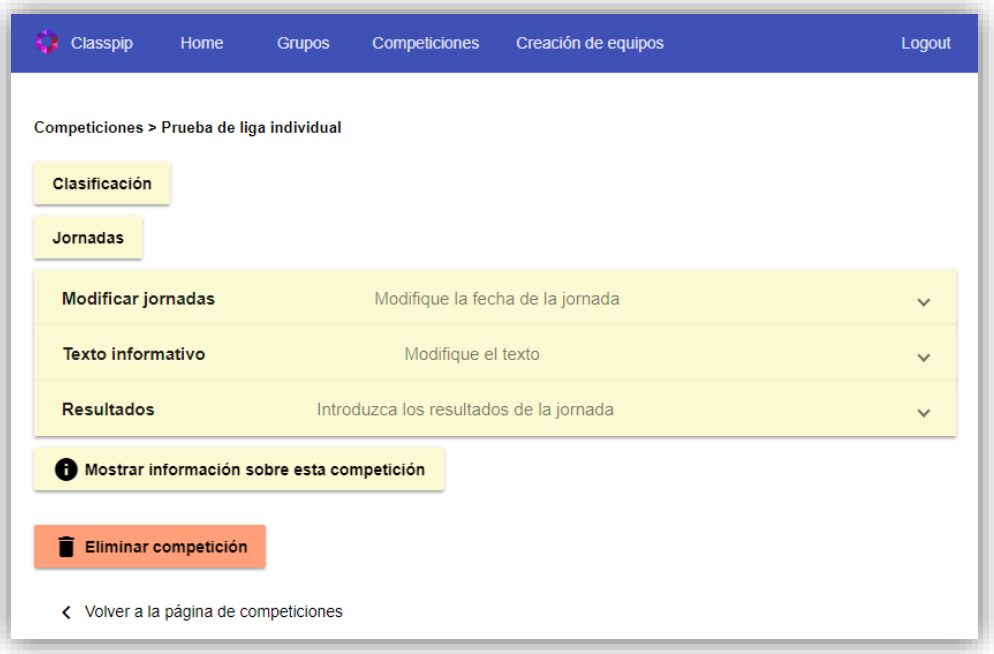

Figura 9.12. Menú principal de la liga modo individual (profesores)

En la **Figura 9.13** y la **Figura 9.14** se observa este menú para competiciones con y sin equipos si accedes como alumno.

Figura 9.13. Menú principal de la liga modo equipos (estudiantes)

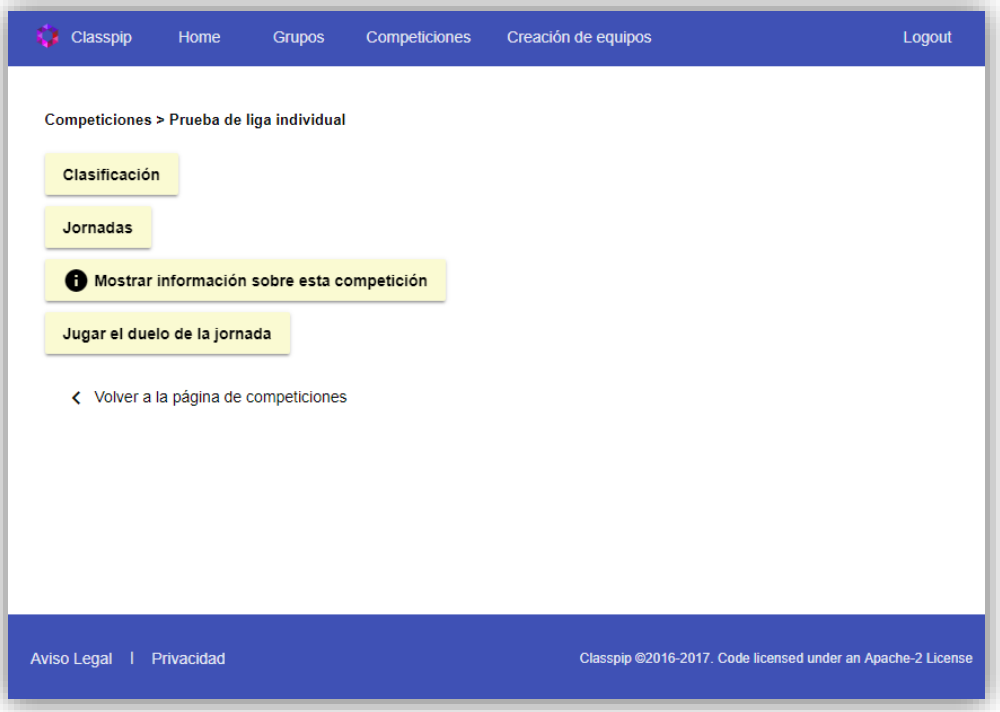

Figura 9.14. Menú principal de la liga modo individual (estudiantes)

9.1.3.2. Clasificación

El aspecto de esta página puede ser diferente por dos motivos:

1. Según cuál sea el modo de la competición (individual o por equipos).
2. Si el número de participantes es par o impar.

Por tanto, en la **Figura 9.15** se observa una competición individual con un número de estudiantes par.

Figura 9.15. Clasificación de la liga individual (número de alumnos par)

En cambio, en la **Figura 9.16**, se puede ver el caso de una competición por equipos cuando el número de éstos es impar.

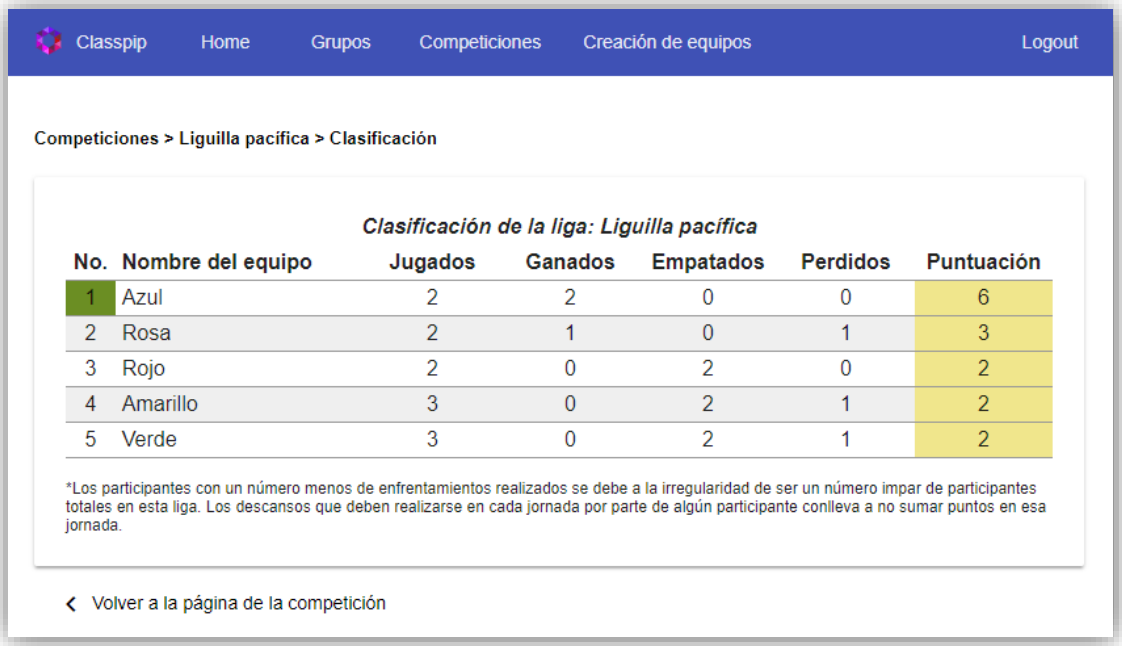

Figura 9.16. Clasificación de la liga por equipos (número de equipos impar)

9.1.3.3. *Jornadas*

Figura 9.17. Calendario de la liga

9.1.3.4. *Introducción de los resultados*

Figura 9.18. Introducción de resultados de la liga (manualmente)

9.1.4. Competición de tipo: Torneo de tenis de doble eliminación

En esta sección del anexo se recopilan todas las apariencias posibles del torneo de tenis de doble eliminación, exceptuando las que comparte con la liga.

9.1.4.1. Menú principal

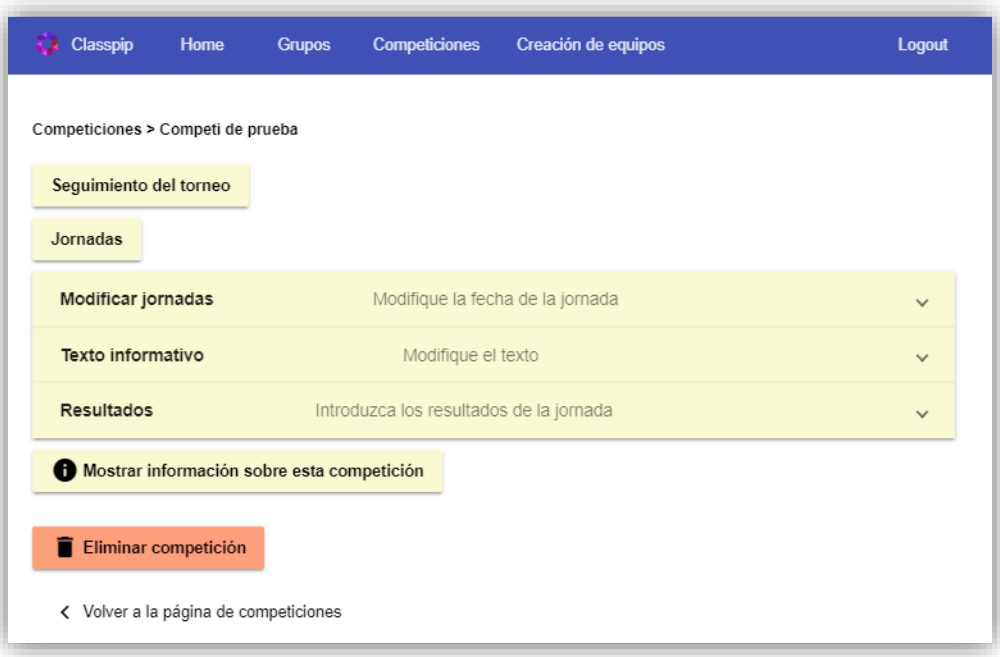

Figura 9.19. Menú principal del torneo de tenis modo individual (profesores)

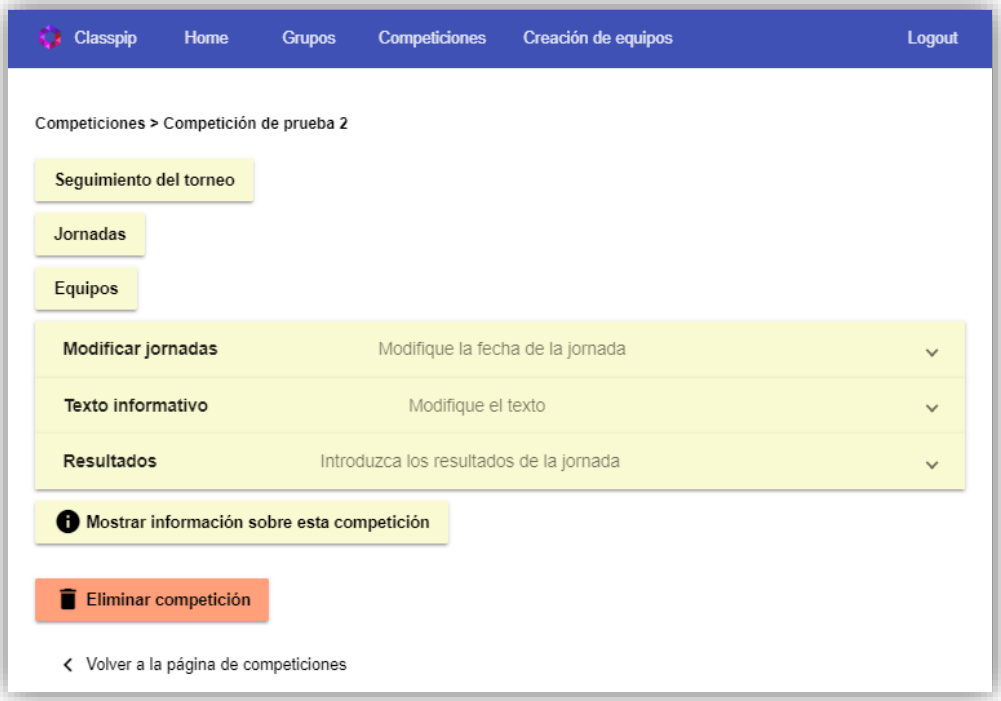

Figura 9.20. Menú principal del torneo de tenis modo equipos (profesores)

Figura 9.22. Menú principal del torneo de tenis modo individual (alumnos)

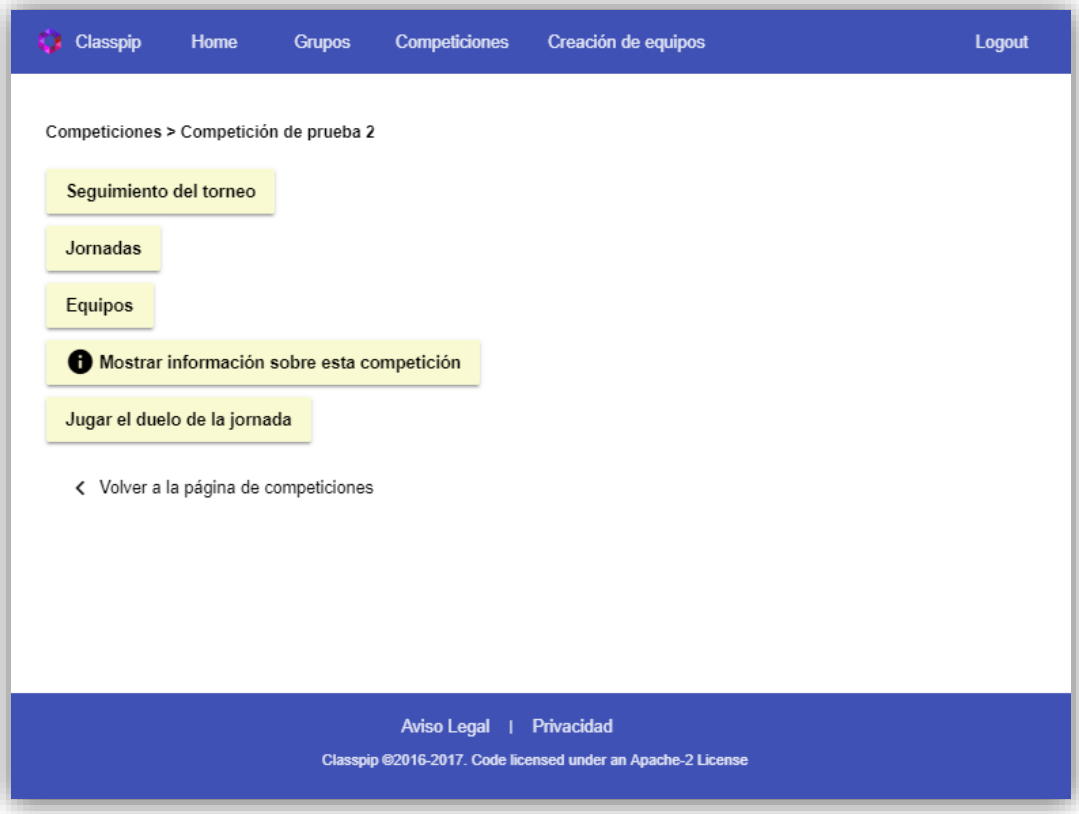

Figura 9.21. Menú principal del torneo de tenis modo equipos (alumnos)

9.1.4.2. Jornadas

La **Figura 9.23** muestra el calendario de una competición de tipo torneo de tenis a modo de equipos habiéndose jugado solamente la primera jornada. A modo individual lo único que cambiaría serían las palabras *Equipo 1* y *Equipo 2* por *Jugador 1* y *Jugador 2*.

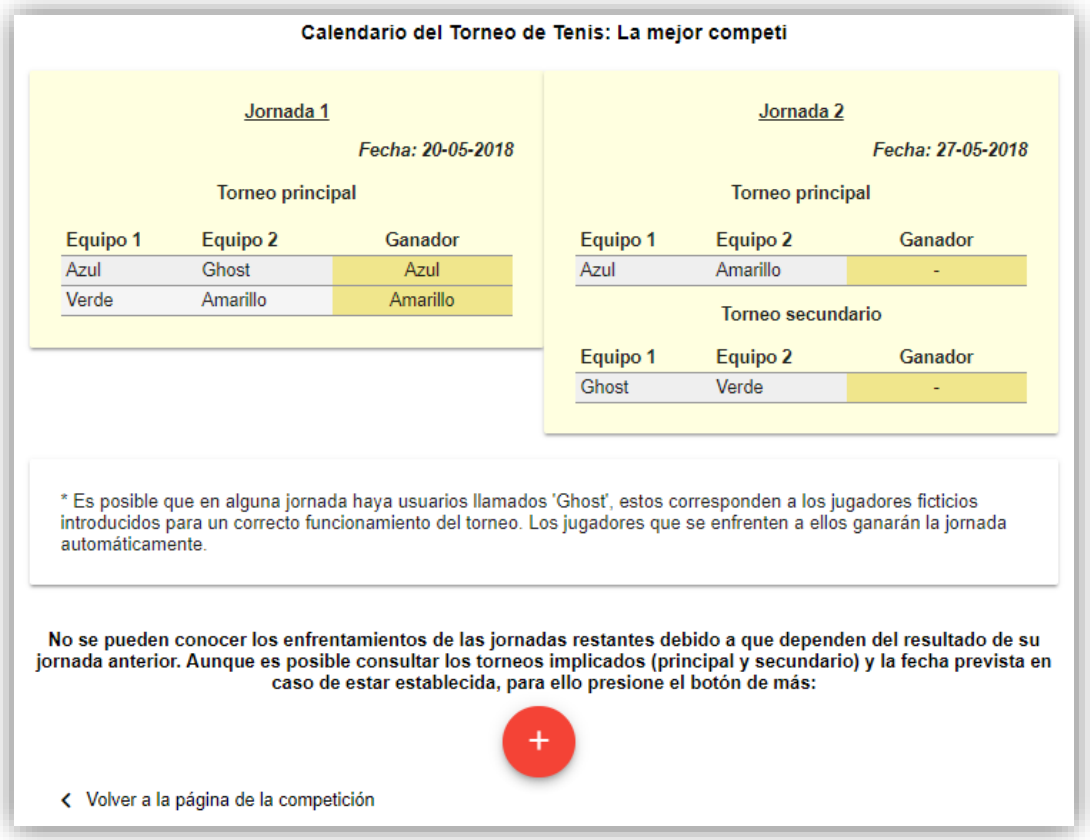

Figura 9.23. Calendario del torneo de tenis con la sección plegada

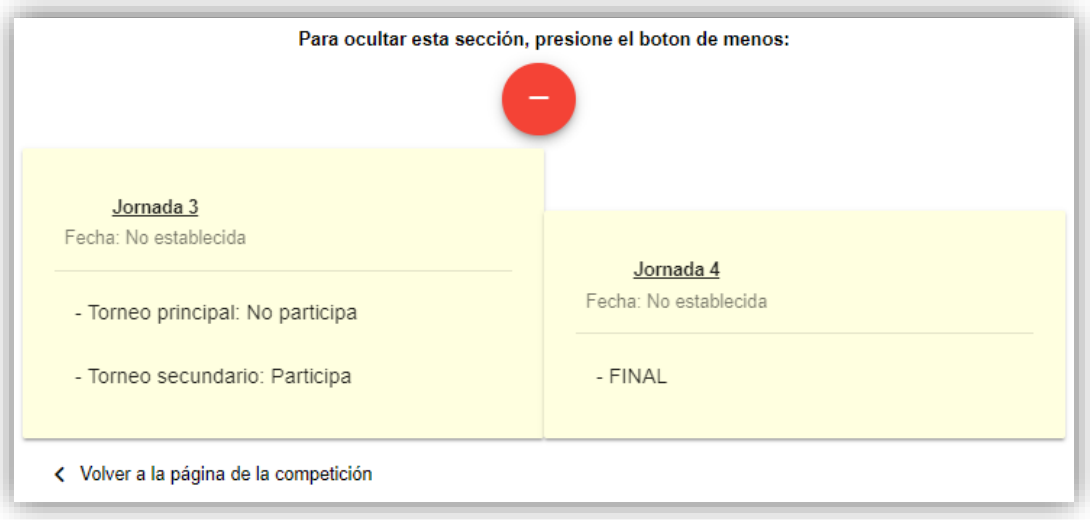

Figura 9.24. Sección plegable del torneo de tenis desplegada

9.1.4.3. Seguimiento del torneo

La herramienta ‘seguimiento del torneo’ irá cambiando de aspecto según de avanzada se encuentre la competición y según la modalidad que sea.

Un ejemplo es la **Figura 9.25**. Se trata de una competición individual en la que aún debe de jugarse la jornada número 2, por tanto, al haberse disputado una sola jornada, los jugadores estarán repartidos entre el torneo principal y el secundario y aún no habrán participantes eliminados. Es a partir de esta jornada desde donde comienzan a sumarse jugadores a la zona de eliminados mientras vayan perdiendo en el torneo secundario.

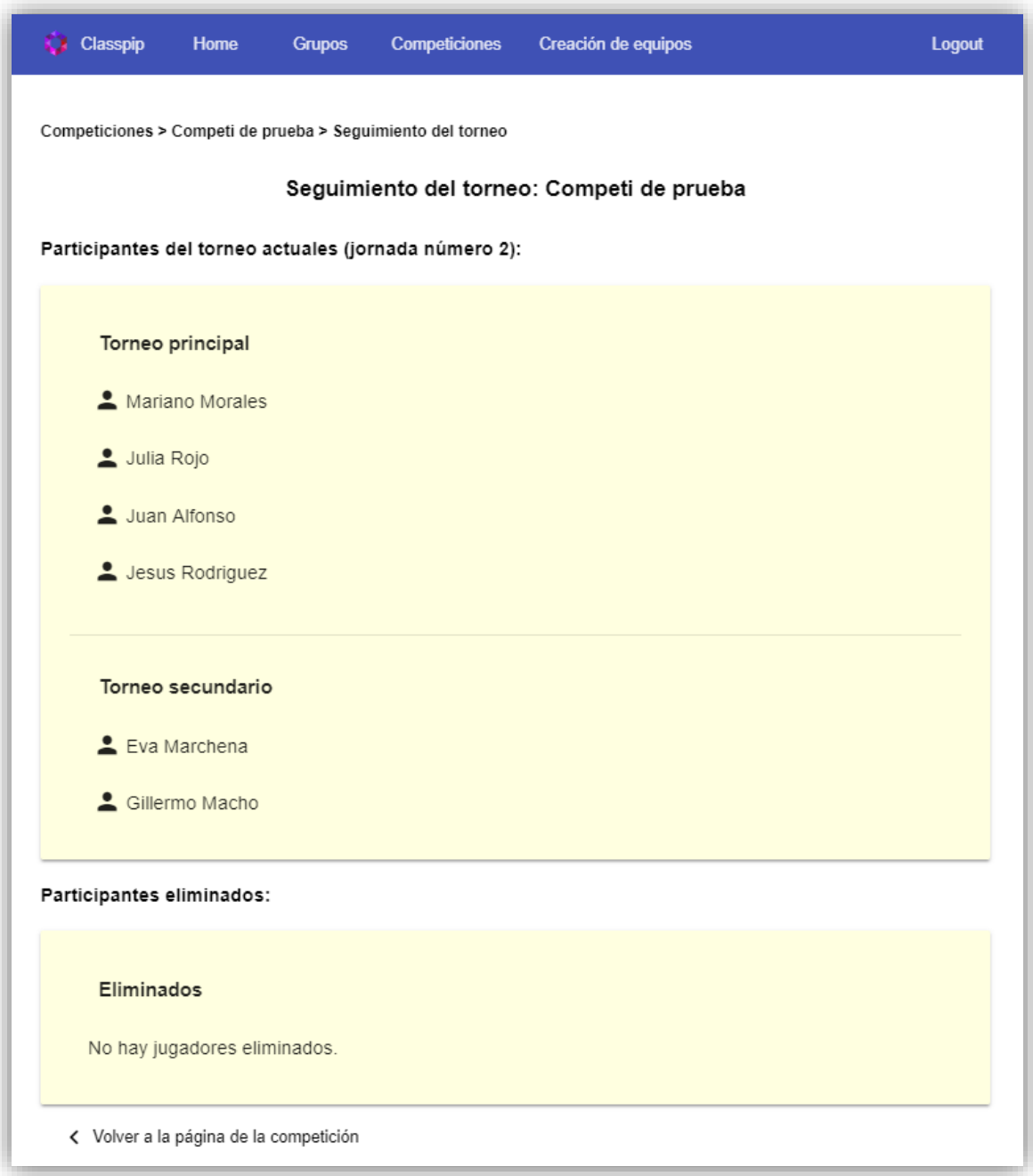

Figura 9.25. Seguimiento del torneo (modo individual)

Si la competición fue a modo equipos y hubiese acabado, el aspecto de ésta se vería como se muestra en la **Figura 9.26**.

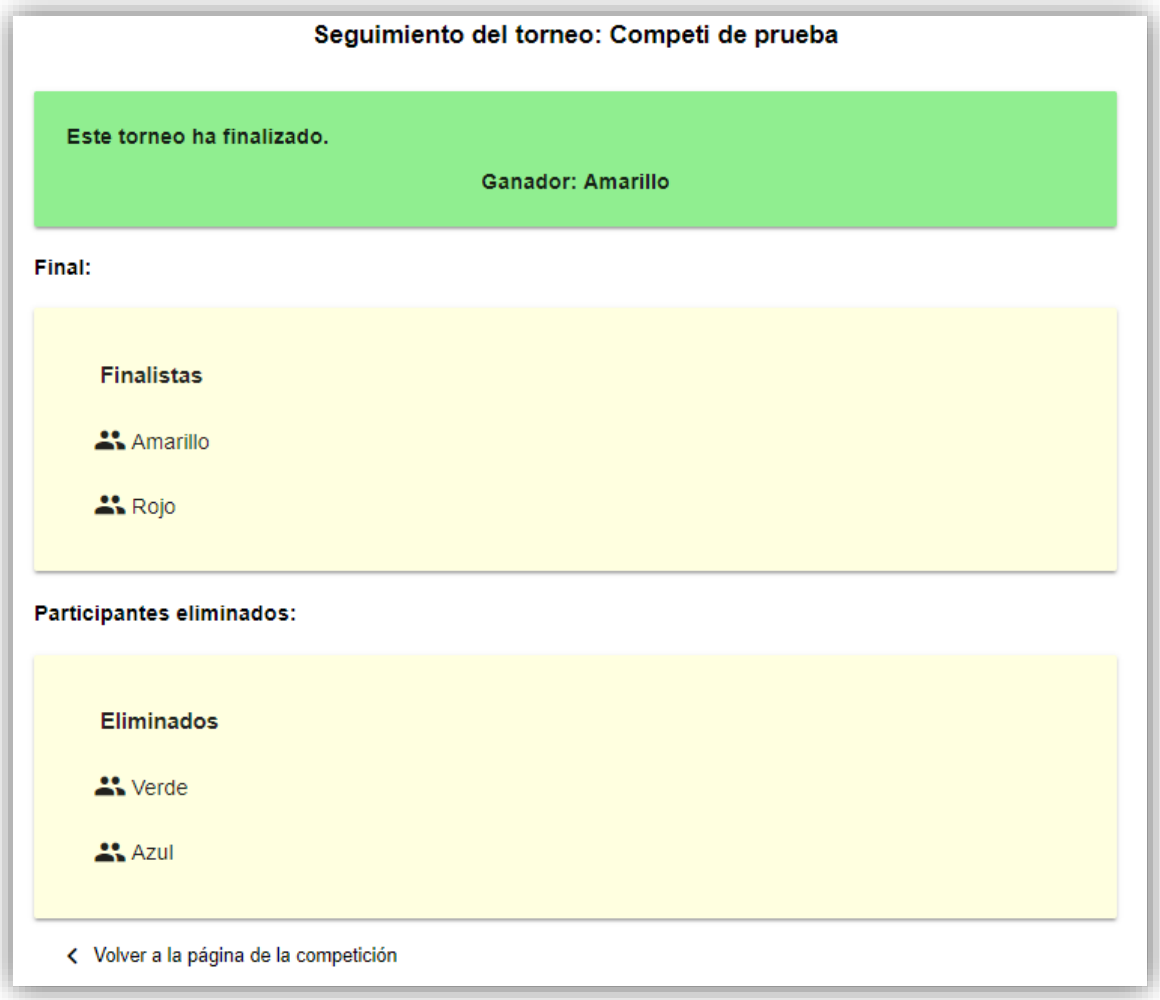

Figura 9.26. Seguimiento del torneo, finalizado (modo equipos)

9.1.4.4. Introducción de resultados

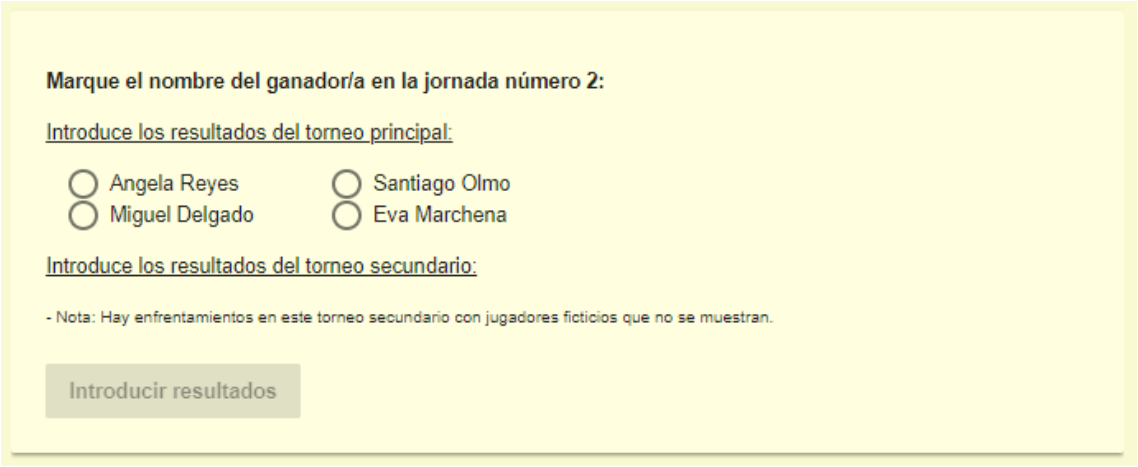

Figura 9.27. Introducción de resultados con jugadores ficticios

La introducción de resultados manual tendrá diferentes aspectos dependiendo de la jornada que toque introducir y de si participan jugadores ficticios.

Puede darse el caso de que participen ambos torneos, como en la **Figura 9.27**, o de que participe uno solo (el principal, como en la **Figura 9.28**, o el secundario). Es posible además que no haya jugadores ficticios y por tanto la aplicación no lo advierta, pero en caso de que haya alguno, ese enfrentamiento no lo muestra y lo comunica en el torneo en donde se encuentre. Esto también puede observarse en estas figuras.

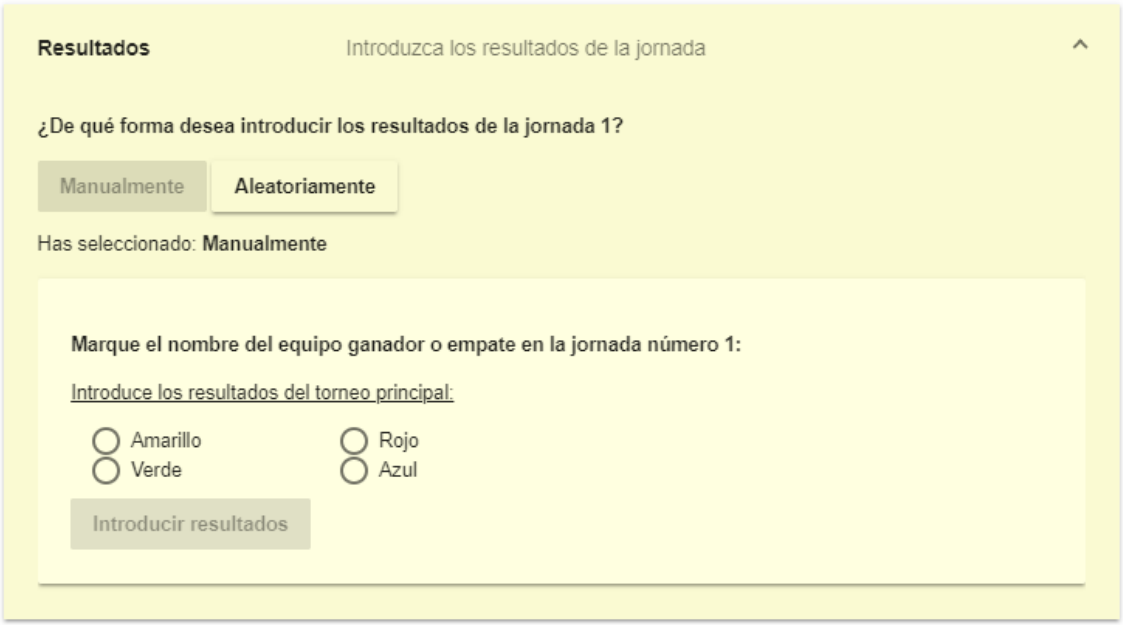

Figura 9.28. Introducción de resultados sin jugadores ficticios

Una vez modificados los resultados, la aplicación revela un mensaje temporal indicando que ha creado los enfrentamientos para la siguiente jornada, **Figura 9.29**, a no ser que se trate de las dos últimas jornadas, en cuyo caso solo habrá un enfrentamiento, **Figura 9.30**.

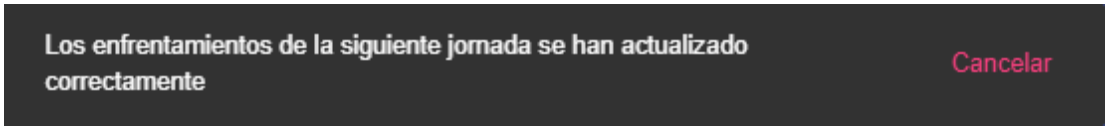

Figura 9.29. Enfrentamientos de la siguiente jornada actualizados

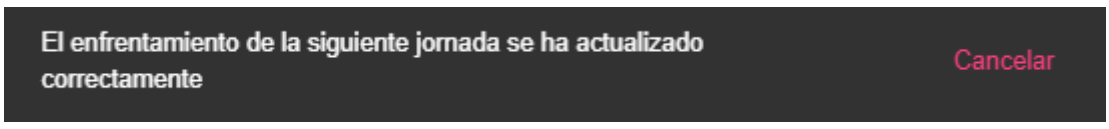

Figura 9.30. Enfrentamiento de la siguiente jornada actualizado.

Al introducir los resultados de la final, en la página no aparecerá ninguno de estos mensajes ya que no habrá más enfrentamientos que disputar.

9.1.5. Secciones compartidas entre la liga y el torneo de tenis

En este apartado se recogen todos los aspectos que son completamente iguales para ambos tipos de competición.

9.1.5.1. Equipos

Figura 9.31. Página de equipos

9.1.5.2. Modificación de jornadas

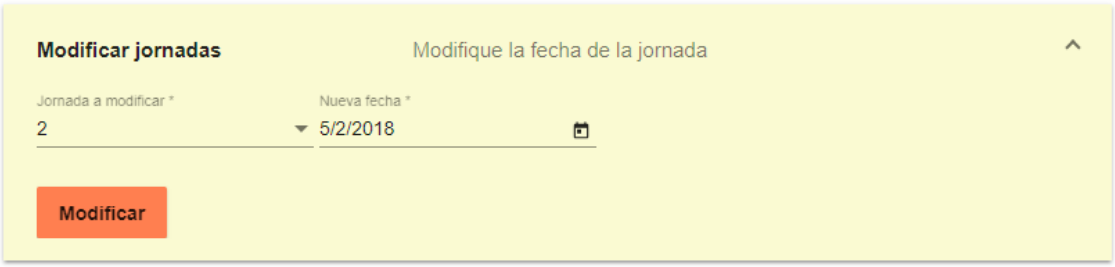

Figura 9.32. Modificación de jornadas

La **Figura 9.32** muestra la herramienta para modificar jornadas, con la cual, una vez modificada alguna de ellas, aparecerá un mensaje temporal de información para reportar que la actualización se ha realizado correctamente, **Figura 9.33**.

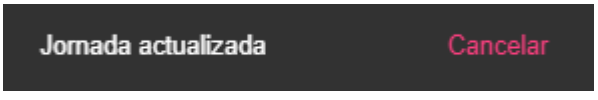

Figura 9.33. Jornada actualizada con éxito

En cambio, en la **Figura 9.34** se observa el mensaje que muestra esta herramienta al no poder modificar ninguna jornada.

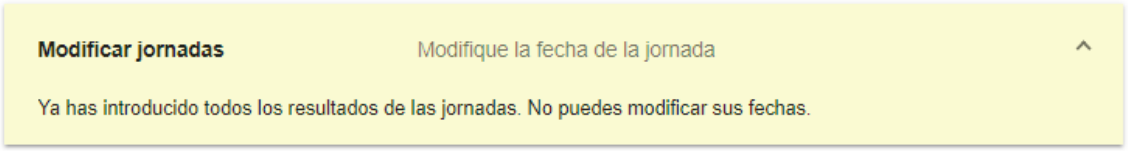

Figura 9.34. Aviso para prohibir la modificación de jornadas.

9.1.5.3. Introducción de resultados

La sección de introducir los resultados manualmente es ligeramente diferente para ambos tipos de competición. Sin embargo, si deseas realizarlo de forma aleatoria, la apariencia será igual, **Figura 9.35**.

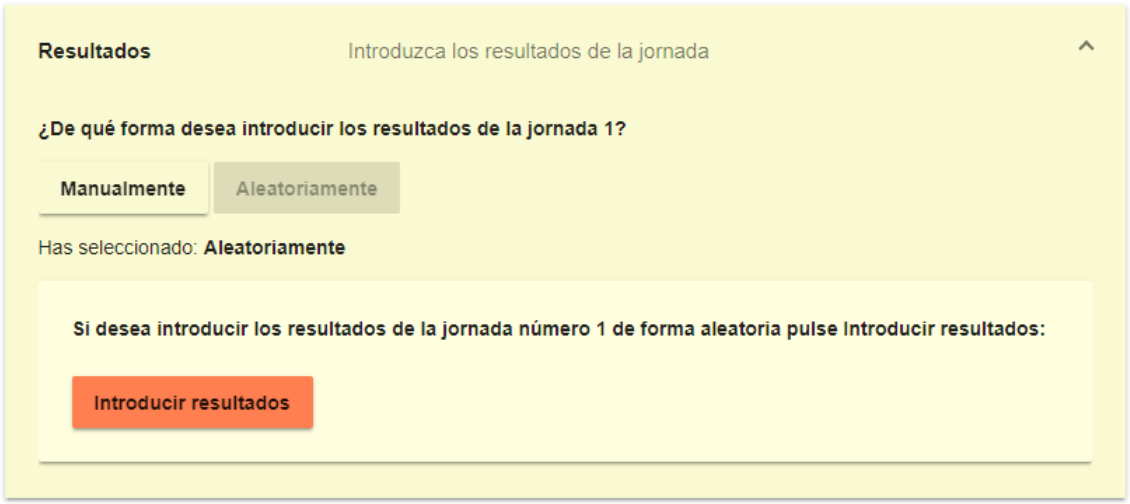

Figura 9.35. Introducción de resultados de la liga (aleatoriamente)

Además, comparten el mensaje que muestra la aplicación al introducir los resultados, **Figura 9.36**.

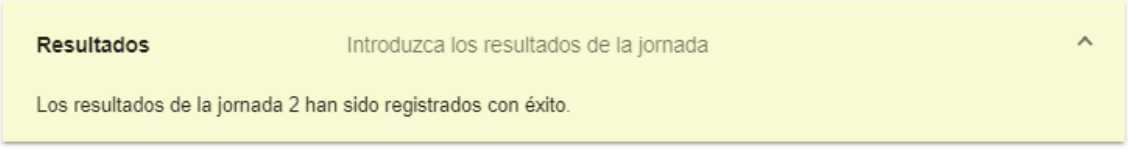

Figura 9.36. Resultados de la jornada registrados con éxito

Una vez se han introducido los resultados de todas las jornadas, es decir, la competición ha finalizado, el mensaje que muestra esta herramienta es el mismo (véase **Figura 9.37**).

La **Figura 9.38** consiste en el mensaje temporal que aparece en la página una vez los resultados han sido registrados.

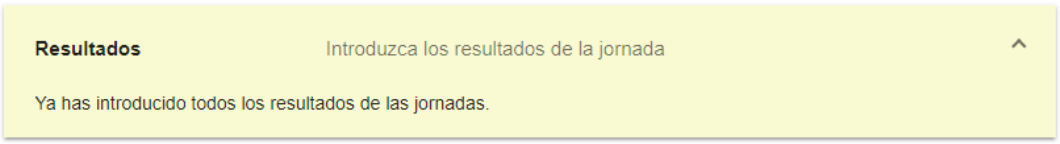

Figura 9.37. Resultados introducidos

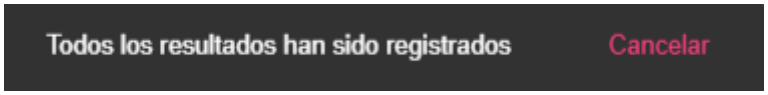

Figura 9.38. Resultados registrados con éxito

9.1.5.4. Modificación del texto informativo

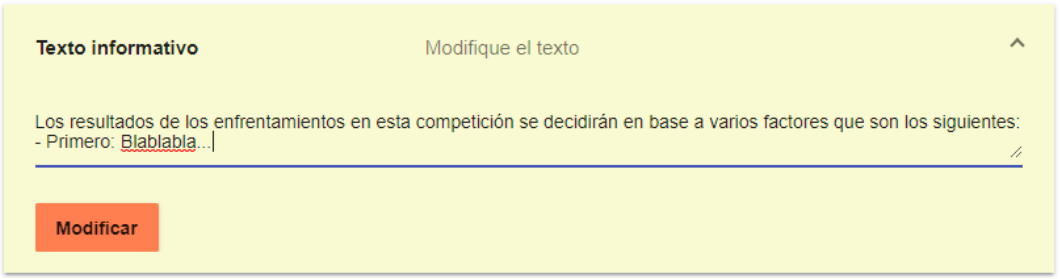

Figura 9.39. Modificación del texto informativo

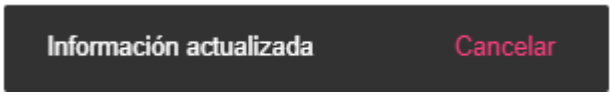

Figura 9.40. Información actualizada con éxito

9.1.5.5. Visualización del texto informativo

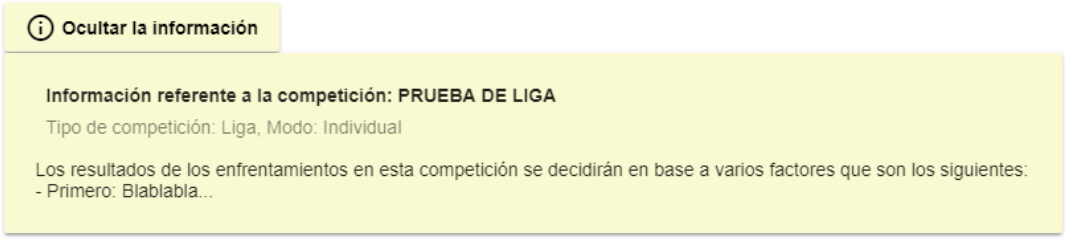

Figura 9.41. Visualización de la información (pestaña desplegada)

9.1.5.6. Eliminación de la competición

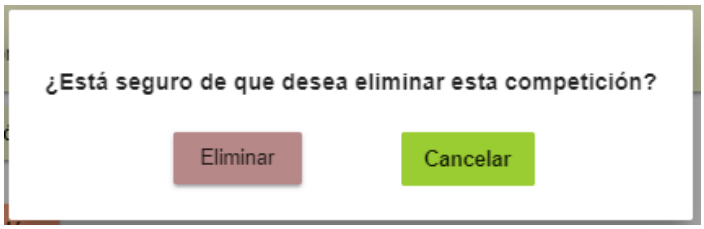

Figura 9.42. Advertencia de eliminar competición

9.2. Imágenes de la sección de creación de equipos en el dashboard

En esta sección del anexo se recopilan las imágenes del aspecto que tiene en la aplicación del panel de administración la creación de equipos.

La **Figura 9.43**. Creación de equipos (inicialización) muestra la primera parte del proceso, que consiste en la selección del grupo y el nombramiento de tantos equipos como se deseen crear.

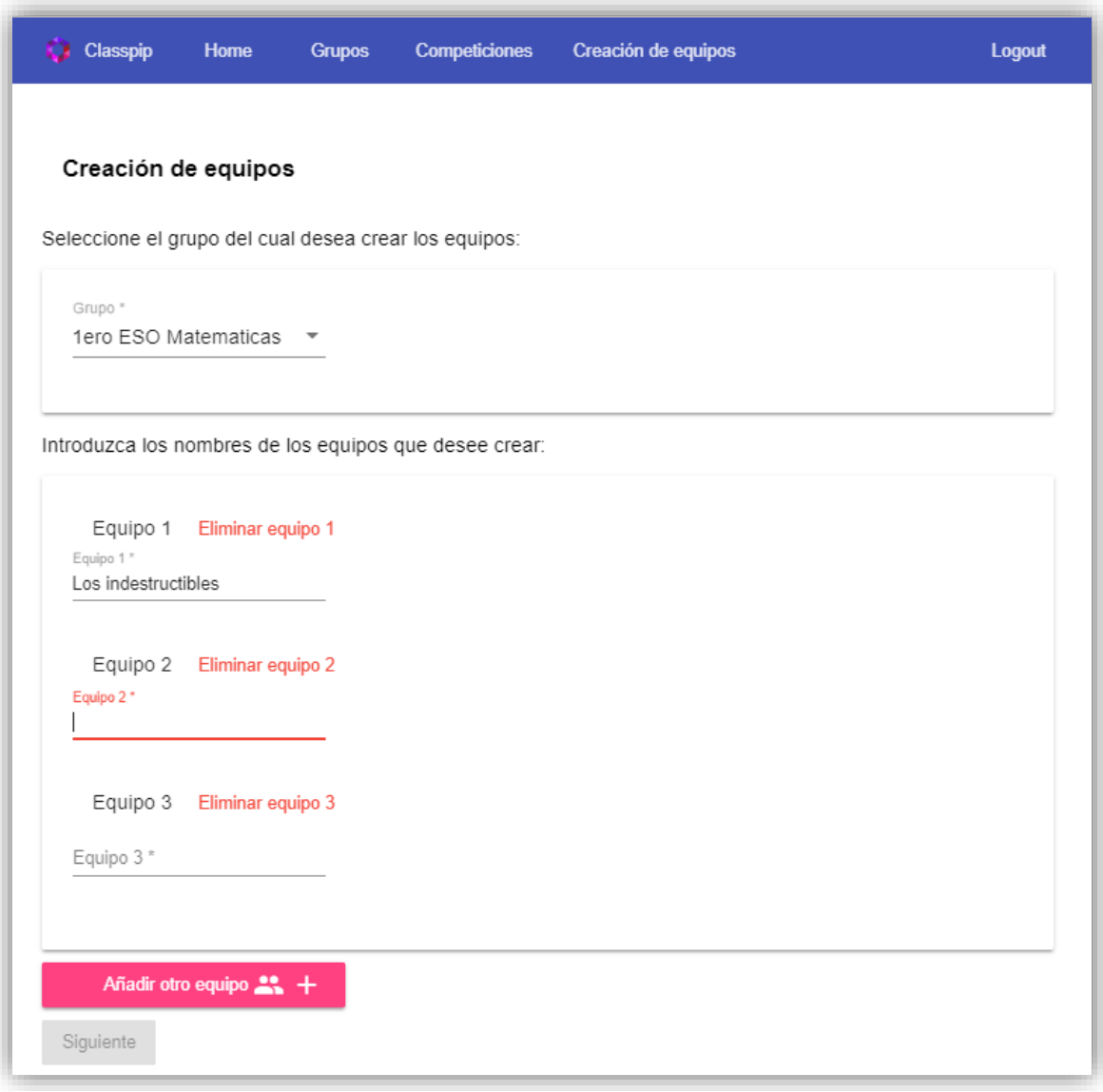

Figura 9.43. Creación de equipos (inicialización)

Esta inicialización es obligatoria para que en este segundo paso, **Figura 9.44**, se muestren los estudiantes que forman parte del grupo seleccionado y puedan distribuirse entre los equipos creados.

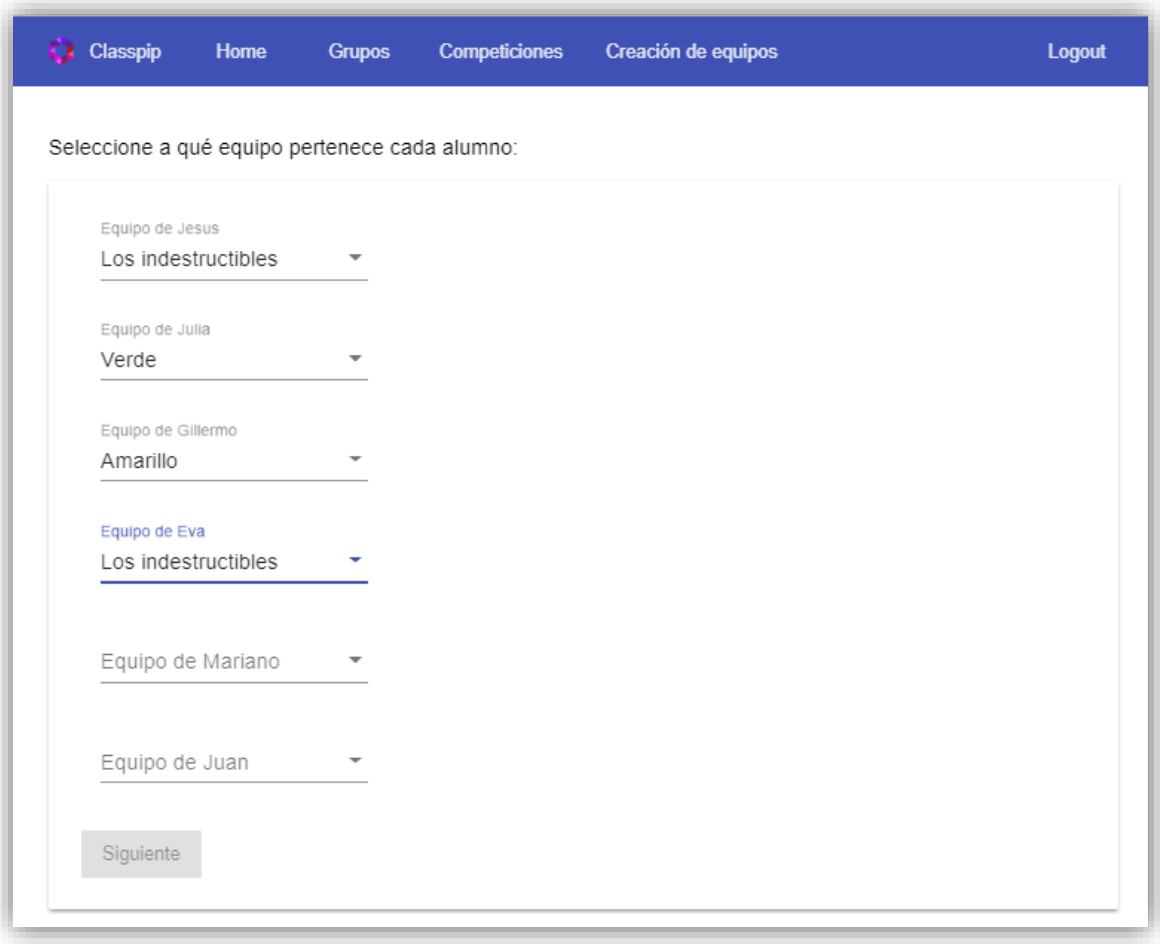

Figura 9.44. Creación de equipos (distribución de participantes)

Por último, una vez completados ambos procesos, se muestra el contenido de la **Figura 9.45**.

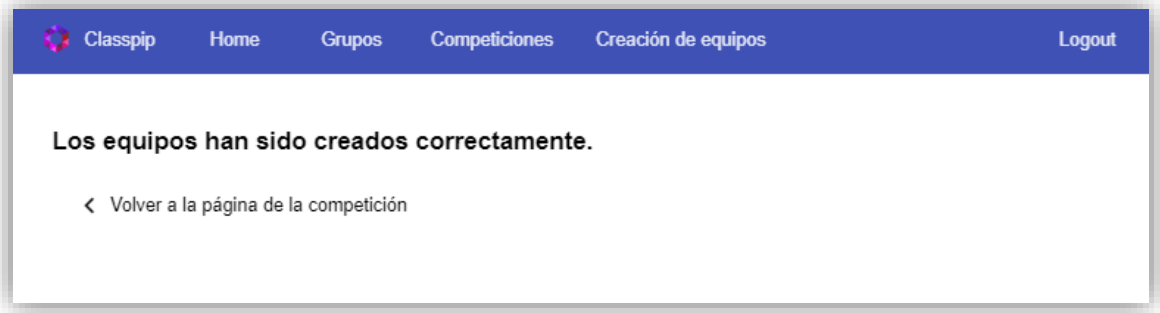

Figura 9.45. Creación de equipos (equipos creados)

9.3. Imágenes de la sección de competiciones en la aplicación móvil

En esta sección del anexo se recopilan las imágenes del aspecto que tiene en la aplicación móvil esta sección de competiciones añadida.

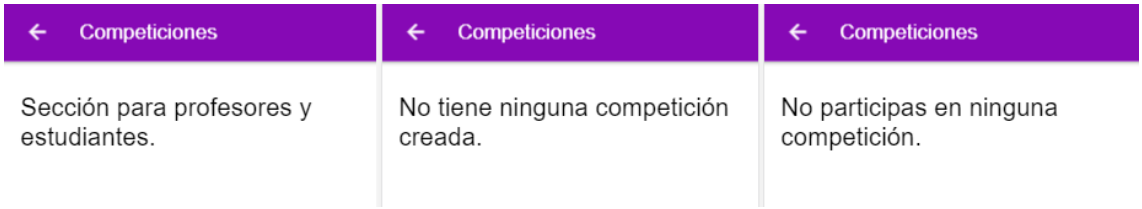

Figura 9.46. Menú principal sin competiciones (director, profesor y estudiante, respectivamente)

El menú principal de esta sección de competiciones muestra la lista de competiciones en la que participan tanto profesores como alumnos. El icono de competición divergerá según se trate de una liga o de un torneo de tenis, **Figura 9.47**.

La **Figura 9.48** muestra a su izquierda la apariencia de un torneo de tenis individual con la información mostrada. A su derecha, se ve el ejemplo de una liga por equipos con la información ocultada.

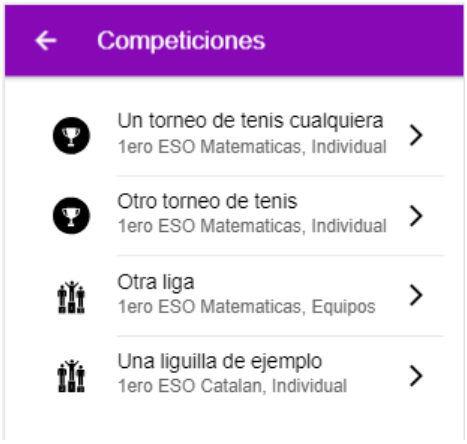

Figura 9.47. Menú principal de la sección de competiciones

Figura 9.48. Menú principal de un torneo de tenis y de una liga

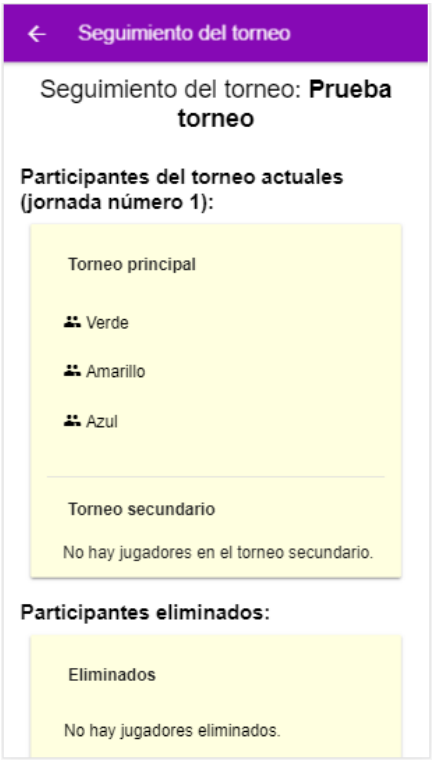

Figura 9.50. Seguimiento del torneo (jornada 1)

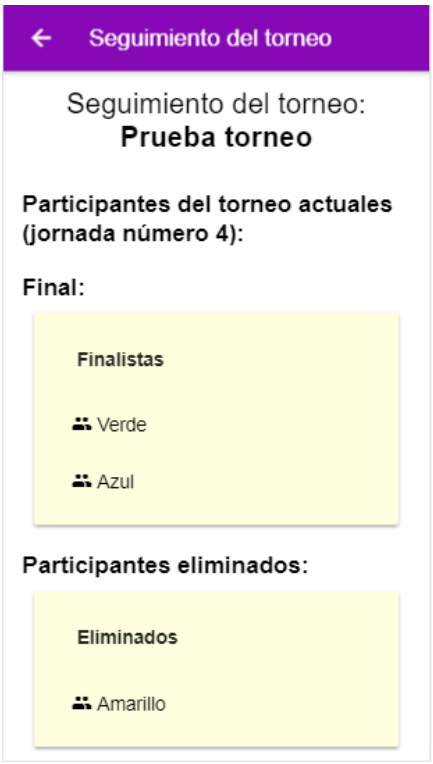

Figura 9.49. Seguimiento del torneo (última jornada)

Figura 9.52. Seguimiento del torneo (jornada 1)

Figura 9.51. Clasificación de la liga

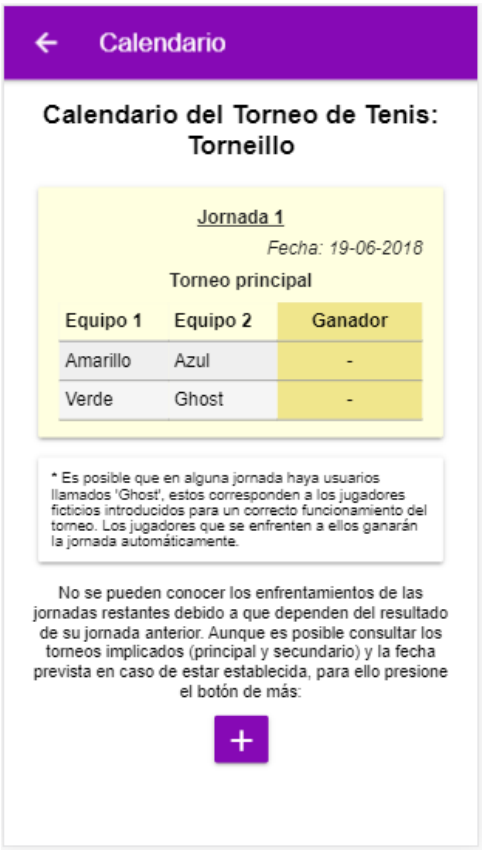

Figura 9.53. Calendario torneo de tenis (sección 1)

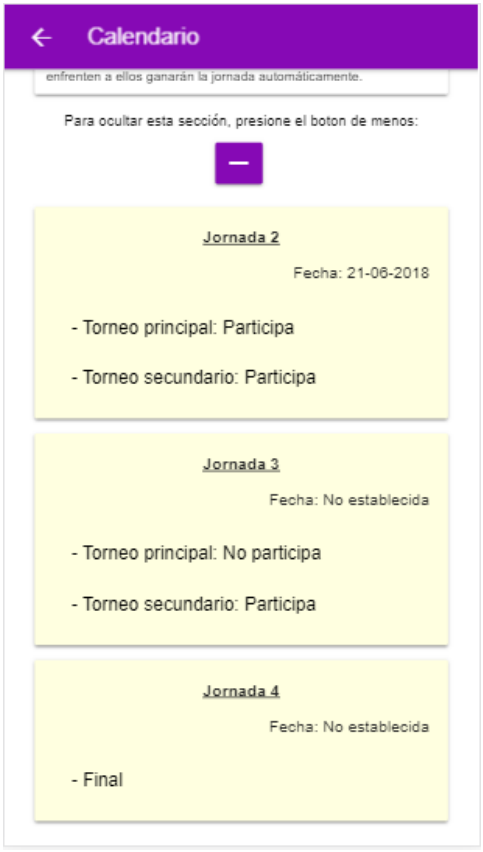

Figura 9.54. Calendario torneo de tenis (sección 2, desplegable)

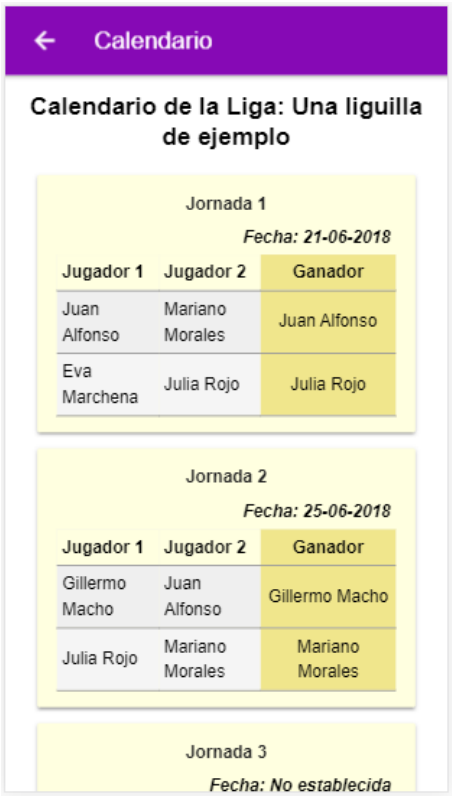

Figura 9.56. Calendario de liga

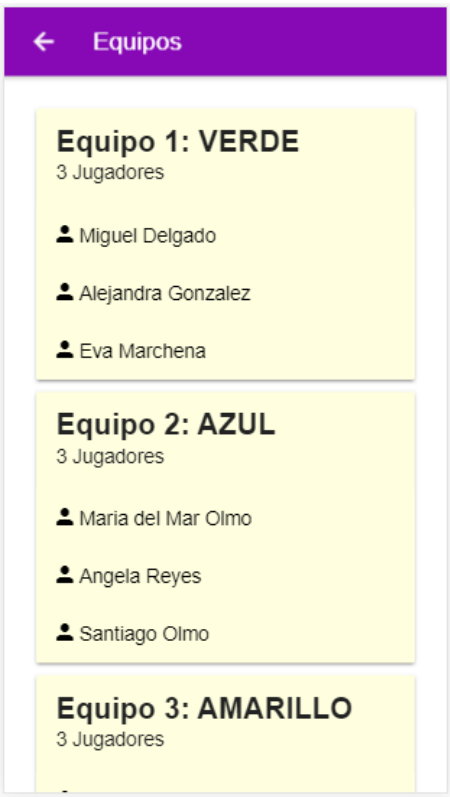

Figura 9.55. Sección equipos

9.4. Servidor API

9.4.1. Relaciones N:N

En la **Figura 9.57** se recogen todos los modelos implicados en este proyecto relacionados mediante hasAndBelongsToMany.

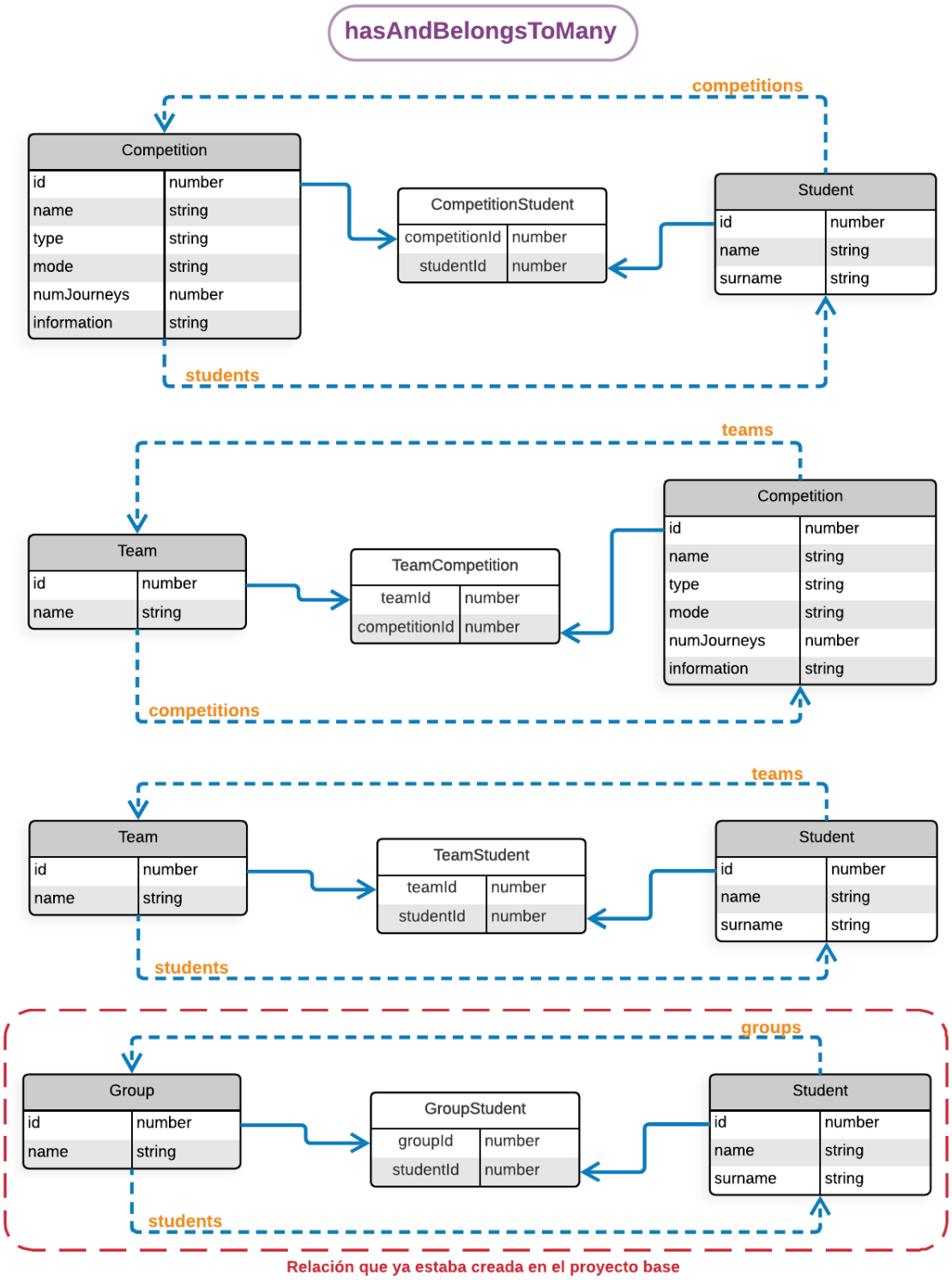

Figura 9.57. Modelos hasAndBelongsToMany

9.4.2. Relaciones 1:N

En la **Figura 9.58** se pueden ver las relaciones HasMany y BelongsTo que se han introducido en este proyecto.

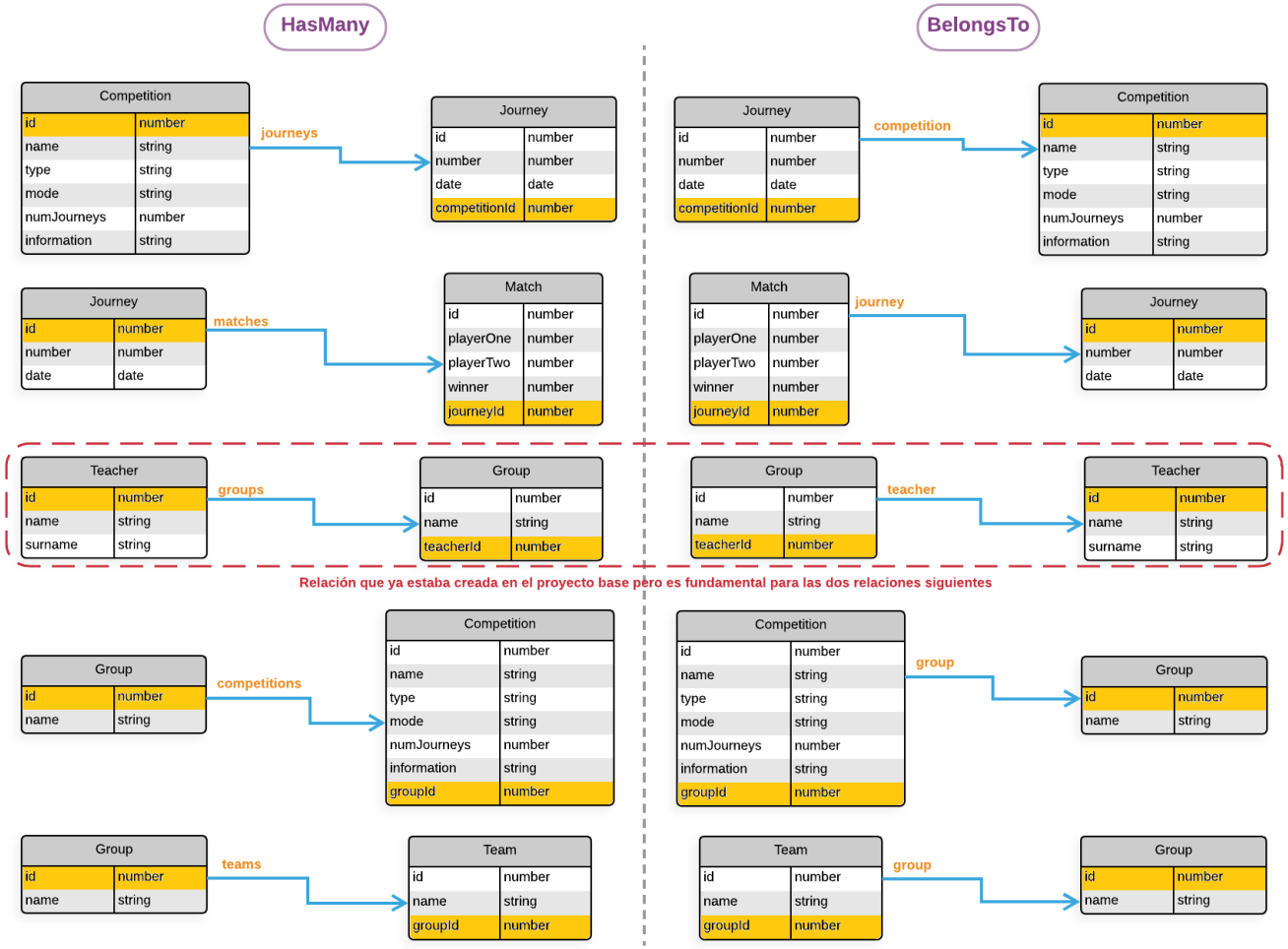

Figura 9.58. Modelos hasMany y belongsTo

9.4.3. Diagrama entidad-relación

En la **Figura 9.59** se muestra un diagrama entidad-relación de los modelos creados en el servidor para la base de datos con una notación de pata de gallo debido a su estilo intuitivo. Además, se exhibe tanto la cardinalidad máxima como la mínima para una mejor comprensión.

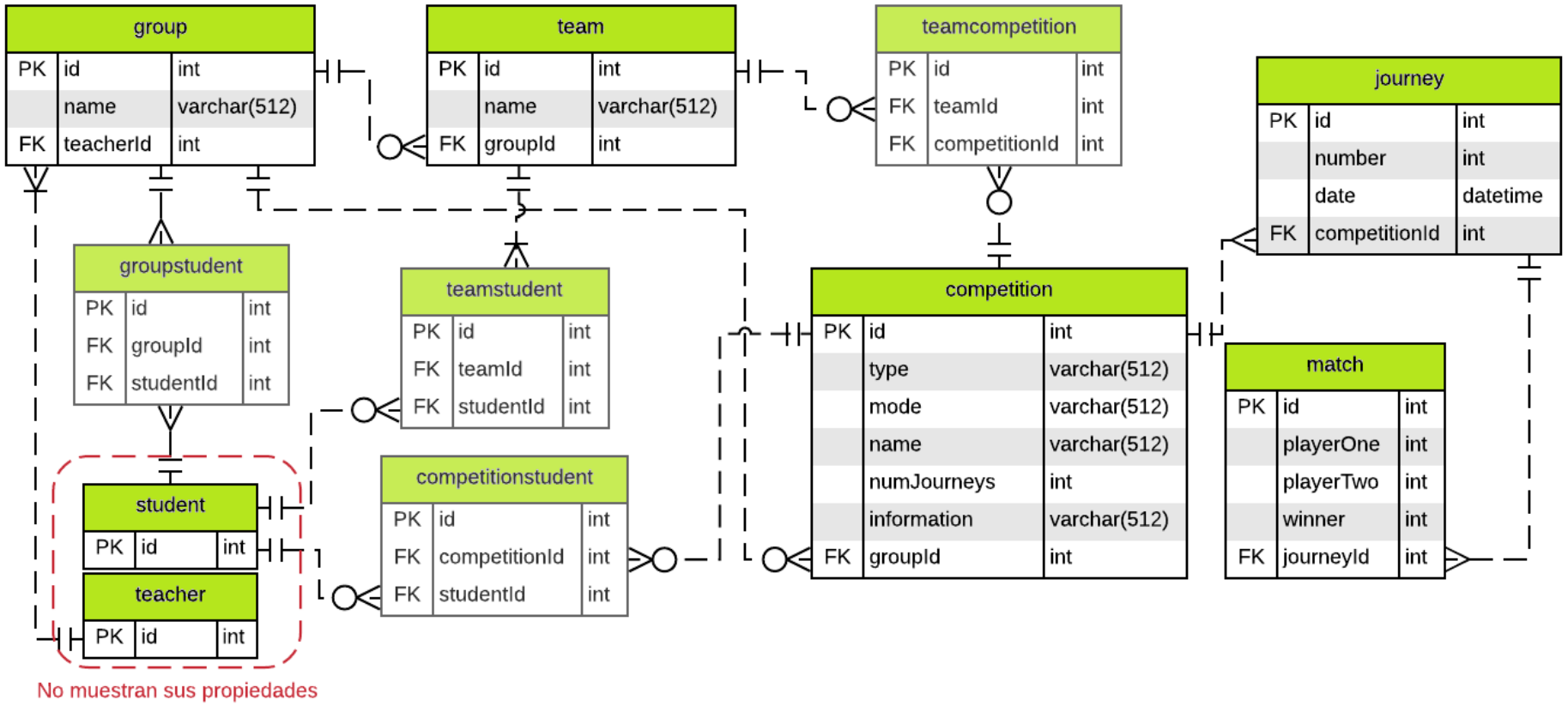

Figura 9.59. Diagrama entidad-relación de los modelos

9.4.4. Modelos del servidor

En esta sección del anexo se recopilan los archivos .json de los modelos creados en el servidor para la base de datos. Solo se incluyen los nuevos modelos que han sido originados en esta versión de la aplicación y podrá verse la relación que han tenido éstos con modelos ya creados anteriormente.

9.4.4.1. Competition.json

```
{
"name": "Competition",
"plural": "competitions",
"base": "PersistedModel",
"idInjection": true,
"options": {
  "validateUpsert": true
},
"properties": {
  "type": {
    "type": "string",
    "required": true
  },
  "mode": {
    "type": "string",
    "required": true
  },
  "name": {
    "type": "string",
    "required": true
  },
  "numJourneys": {
    "type": "number",
    "required": true
  },
  "information": {
    "type": "string",
    "required": false
  }
},
"validations": [],
"relations": {
  "journeys": {
    "type": "hasMany",
    "model": "Journey",
    "foreignKey": "competitionId"
  },
  "students": {
    "type": "hasAndBelongsToMany",
    "model": "Student",
    "foreignKey": "studentId"
  },
  "teams": {
    "type": "hasAndBelongsToMany",
    "model": "Team",
    "foreignKey": "teamId"
  },
  "group": {
    "type": "belongsTo",
    "model": "Group",
    "foreignKey": "groupId"
  }
},
"acls": [
  {
    "accessType": "*",
    "principalType": "ROLE",
    "principalId": "$everyone",
    "permission": "DENY"
  },
  {
```

```
      "accessType": "READ",
      "principalType": "ROLE",
      "principalId": "$authenticated",
      "permission": "ALLOW"
    },
    {
      "accessType": "WRITE",
      "principalType": "ROLE",
      "principalId": "$authenticated",
      "permission": "ALLOW"
    }
  ],
  "methods": {}
}
```

9.4.4.2. Journey.json

```
{
  "name": "Journey",
  "plural": "journeys",
  "base": "PersistedModel",
  "idInjection": true,
  "options": {
    "validateUpsert": true
  },
  "properties": {
    "number": {
      "type": "number",
      "required": true
    },
    "date": {
      "type": "date",
      "required": false
    }
  },
  "validations": [],
  "relations": {
    "competition": {
      "type": "belongsTo",
      "model": "Competition",
      "foreignKey": "competitionId"
    },
    "matches": {
      "type": "hasMany",
      "model": "Match",
      "foreignKey": "journeyId"
    }
  },
  "acls": [
    {
      "accessType": "*",
      "principalType": "ROLE",
      "principalId": "$everyone",
      "permission": "DENY"
    },
    {
      "accessType": "READ",
      "principalType": "ROLE",
      "principalId": "$authenticated",
      "permission": "ALLOW"
    },
    {
      "accessType": "WRITE",
      "principalType": "ROLE",
      "principalId": "$authenticated",
      "permission": "ALLOW"
    }
  ],
  "methods": {}
}
```

9.4.4.3. Match.json

```
{
  "name": "Match",
  "plural": "matches",
  "base": "PersistedModel",
  "idInjection": true,
  "options": {
    "validateUpsert": true
  },
  "properties": {
    "playerOne": {
      "type": "number",
      "required": true
    },
    "playerTwo": {
      "type": "number",
      "required": true
    },
    "winner": {
      "type": "number",
      "required": false,
      "default": 0
    }
  },
  "validations": [],
  "relations": {
    "journey": {
      "type": "belongsTo",
      "model": "Journey",
      "foreignKey": "journeyId"
    }
  },
  "acls": [
    {
      "accessType": "*",
      "principalType": "ROLE",
      "principalId": "$everyone",
      "permission": "DENY"
    },
    {
      "accessType": "READ",
      "principalType": "ROLE",
      "principalId": "$authenticated",
      "permission": "ALLOW"
    },
    {
      "accessType": "WRITE",
      "principalType": "ROLE",
      "principalId": "$authenticated",
      "permission": "ALLOW"
    }
  ],
  "methods": {}
}
```

9.4.4.4. Team.json

```
{
  "name": "Team",
  "plural": "teams",
  "base": "PersistedModel",
  "idInjection": true,
  "options": {
    "validateUpsert": true
  },
  "properties": {
    "name": {
      "type": "string", "required": true
    }
```

```
  },
  "validations": [],
  "relations": {
    "competitions": {
      "type": "hasAndBelongsToMany",
      "model": "Competition",
      "foreignKey": "competitionId"
    },
    "students": {
      "type": "hasAndBelongsToMany",
      "model": "Student",
      "foreignKey": "studentId"
    },
    "group": {
      "type": "belongsTo",
      "model": "Group",
      "foreignKey": "groupId"
    }
  },
  "acls": [
    {
      "accessType": "*",
      "principalType": "ROLE",
      "principalId": "$everyone",
      "permission": "DENY"
    },
    {
      "accessType": "READ",
      "principalType": "ROLE",
      "principalId": "$authenticated",
      "permission": "ALLOW"
    },
    {
      "accessType": "WRITE",
      "principalType": "ROLE",
      "principalId": "$authenticated",
      "permission": "ALLOW"
    }
  ],
  "methods": {}
}
```

9.4.5. Boot script

Fragmento de código para crear equipos en: 06-create-students.js y agregarles estudiantes.

```
app.models.Team.create([{
  id: 10, name: 'Manzana', groupId: 2
 }, {
  id: 11, name: 'Naranja', groupId: 2
 }], function (err, teams) {
  teams[0].students.add(students[2], function (err) {
   if (err) throw err;
    teams[0].students.add(students[3], function (err) {
     if (err) throw err;
      teams[0].students.add(students[4], function (err) {
       if (err) throw err;
        teams[1].students.add(students[5], function (err) {
         if (err) throw err;
          teams[1].students.add(students[6], function (err) {
           if (err) throw err;
            teams[1].students.add(students[12], function (err) {
             if (err) throw err;
             process.nextTick(cb);
            })
          })
        })
      })
    })
  })
});
```

9.5. Modelos en el panel de administración

9.5.1. Clase competición

```
export class Competition {

  private _id: string;
  private  name: string;
  private  type: string;
  private _mode: string;
  private _numParticipants: number;
  private _numJourneys: number;
  private  information: string;
  private  groupId: number;
  private _grade: Grade;
  private _matter: Matter;

  constructor(id?: string, name?: string, type?: string, mode?: string,
    numParticipants?: number, numJourneys?: number, information?: string,
    groupId?: number, grade?: Grade, matter?: Matter) {
    this._id = id;
    this._name = name;
    this. type = type;
    this. mode = mode;
    this. numParticipants = numParticipants;
    this. numJourneys = numJourneys;
    this._information = information;
    this._groupId = groupId;
    this. grade = grade;
    this. matter = matter;
  }

  /* tslint:disable */
  static toObject(object: any): Competition {
    /* tslint:enable */
    const result: Competition = new Competition();
    if (object != null) {
      result.id = object.id;
      result.name = object.name;
      result.type = object.type;
      result.mode = object.mode;
      result.numParticipants = object.numParticipants;
      result.numJourneys = object.numJourneys;
      result.information = object.information;
      result.groupId = object.groupId;
    }
    return result;
  }

  /* tslint:disable */
  static toObjectArray(object: any): Array<Competition> {
    /* tslint:enable */
    const resultArray: Array<Competition> = new Array<Competition>();
    if (object != null) {
      for (let i = 0; i < object.length; i++) {
        resultArray.push(Competition.toObject(object[i]));
      }
    }
    return resultArray;
  }

  /*  Getters and Setters  */

  public get id(): string {
    return this._id;
  }

  public set id(value: string) {
    this. id = value;
  }
```

```
  public get name(): string {
    return this._name;
  }

  public set name(value: string) {
    this. name = value;
  }

  public get type(): string {
    return this._type;
  }

  public set type(value: string) {
    this._type = value;
  }

  public get mode(): string {
    return this. mode;
  }

  public set mode(value: string) {
    this. mode = value;
  }

  public get numParticipants(): number {
    return this._numParticipants;
  }

  public set numParticipants(value: number) {
    this._numParticipants = value;
  }

  public get numJourneys(): number {
    return this. numJourneys;
  }

  public set numJourneys(value: number) {
    this._numJourneys = value;
  }

  public get information(): string {
    return this._information;
  }

  public set information(value: string) {
    this. information = value;
  }

  public get groupId(): number {
    return this. groupId;
  }

  public set groupId(value: number) {
    this._groupId = value;
  }

  public get grade(): Grade {
    return this. grade;
  }

  public set grade(value: Grade) {
    this. grade = value;
  }

  public get matter(): Matter {
    return this._matter;
  }

  public set matter(value: Matter) {
    this._matter = value;
  }
}
```

9.5.2. Clase jornada

```
export class Journey {

  private  id: string;
  private _number: number;
  private _date: Date;
  private _competitionId: number;
  private  completed: boolean;

  constructor(id?:  string,  number?:  number,  date?:  Date,  competitionId?:  number,
completed?: boolean) {
    this._id = id;
    this._number = number;
    this. date = date;
    this. competitionId = competitionId;
    this. completed = completed;
  }

  /* tslint:disable */
  static toObject(object: any): Journey {
    /* tslint:enable */
    const result: Journey = new Journey();
    if (object != null) {
      result.id = object.id;
      result.number = object.number;
      result.date = object.date;
      result.competitionId = object.competitionId;
    }
    return result;
  }

  /* tslint:disable */
  static toObjectArray(object: any): Array<Journey> {
    /* tslint:enable */
    const resultArray: Array<Journey> = new Array<Journey>();
    if (object != null) {
      for (let i = 0; i < object.length; i++) {
        resultArray.push(Journey.toObject(object[i]));
      }
    }
    return resultArray;
  }


  /*  Getters and Setters  */

  public get id(): string {
    return this. id;
  }

  public set id(value: string) {
    this. id = value;
  }

  public get number(): number {
    return this._number;
  }

  public set number(value: number) {
    this. number = value;
  }

  public get date(): Date {
    return this. date;
  }

  public set date(value: Date) {
    this._date = value;
  }

  public get competitionId(): number {
    return this._competitionId;
  }
```

```
  public set competitionId(value: number) {
    this._competitionId = value;
  }

  public get completed(): boolean {
    return this. completed;
  }

  public set completed(value: boolean) {
    this._completed = value;
  }
}
```

9.5.3. Clase enfrentamiento

```
export class Match {

  private  id: string;
  private  playerOne: number;
  private  playerTwo: number;
  private _winner: number;
  private _journeyId: number;
  private _namePlayerOne: string;
  private  namePlayerTwo: string;
  private  result: string;

  constructor(id?: string, playerOne?: number, playerTwo?: number, winner?: number,
journeyId?: number,
  namePlayerOne?: string, namePlayerTwo?: string, result?: string) {
    this. id = id;
    this. playerOne = playerOne;
    this._playerTwo = playerTwo;
    this._winner = winner;
    this._journeyId = journeyId;
    this. namePlayerOne = namePlayerOne;
    this. namePlayerTwo = namePlayerTwo;
    this._result = result;
  }

  /* tslint:disable */
  static toObject(object: any): Match {
    /* tslint:enable */
    const result: Match = new Match();
    if (object != null) {
      result.id = object.id;
      result.playerOne = object.playerOne;
      result.playerTwo = object.playerTwo;
      result.winner = object.winner;
      result.journeyId = object.journeyId;
    }
    return result;
  }

  /* tslint:disable */
  static toObjectArray(object: any): Array<Match> {
    /* tslint:enable */
    const resultArray: Array<Match> = new Array<Match>();
    if (object != null) {
      for (let i = 0; i < object.length; i++) {
        resultArray.push(Match.toObject(object[i]));
      }
    }
    return resultArray;
  }


  /*  Getters and Setters  */

  public get id(): string {
    return this. id;
  }

  public set id(value: string) {
```

```
    this. id = value;
  }

  public get playerOne(): number {
    return this._playerOne;
  }

  public set playerOne(value: number) {
    this._playerOne = value;
  }

  public get playerTwo(): number {
    return this. playerTwo;
  }

  public set playerTwo(value: number) {
    this._playerTwo = value;
  }

  public get winner(): number {
    return this._winner;
  }

  public set winner(value: number) {
    this._winner = value;
  }

  public get journeyId(): number {
    return this. journeyId;
  }

  public set journeyId(value: number) {
    this. journeyId = value;
  }
  public get namePlayerOne(): string {
    return this._namePlayerOne;
  }

  public set namePlayerOne(value: string) {
    this. namePlayerOne = value;
  }
  public get namePlayerTwo(): string {
    return this._namePlayerTwo;
  }

  public set namePlayerTwo(value: string) {
    this. namePlayerTwo = value;
  }
  public get result(): string {
    return this. result;
  }

  public set result(value: string) {
    this._result = value;
  }
}
```

9.5.4. Clase equipo

```
export class Team {

  private  id: string;
  private  name: string;
  private  numPlayers: number;
  private _groupId: number;

  constructor(id?: string, name?: string, numPlayers?: number, groupId?: number) {
    this. id = id;
    this. name = name;
    this. numPlayers = numPlayers;
    this._groupId = groupId;
  }
```

```
  /* tslint:disable */
  static toObject(object: any): Team {
    /* tslint:enable */
    const result: Team = new Team();
    if (object != null) {
      result.id = object.id;
      result.name = object.name;
      result.numPlayers = object.numPlayers;
      result.groupId = object.groupId;
    }
    return result;
  }

  /* tslint:disable */
  static toObjectArray(object: any): Array<Team> {
    /* tslint:enable */
    const resultArray: Array<Team> = new Array<Team>();
    if (object != null) {
      for (let i = 0; i < object.length; i++) {
        resultArray.push(Team.toObject(object[i]));
      }
    }
    return resultArray;
  }


  /*  Getters and Setters  */

  public get id(): string {
    return this._id;
  }

  public set id(value: string) {
    this. id = value;
  }

  public get name(): string {
    return this._name;
  }

  public set name(value: string) {
    this._name = value;
  }

  public get numPlayers(): number {
    return this. numPlayers;
  }

  public set numPlayers(value: number) {
    this. numPlayers = value;
  }

  public get groupId(): number {
    return this._groupId;
  }

  public set groupId(value: number) {
    this. groupId = value;
  }
}
```

9.6. Interpretación de enfrentamientos en el torneo de tenis

En la **Figura 9.60**. Interpretación de los enfrentamientos (torneo de tenis) se recopila la interpretación de los datos relativos a los enfrentamientos para desarrollar algunas funciones de la aplicación como son:

1. El mecanismo para repartir los enfrentamientos entre los distintos torneos. Útil en la herramienta de introducción de resultados y en el calendario.
2. El mecanismo para construir los enfrentamientos de la siguiente jornada.
3. El mecanismo para elaborar la sección del seguimiento del torneo.

Mecanismo para repartir los enfrentamientos entre los distintos torneos: útil para mostrar cada participante en su correspondiente torneo para la introducción de los resultados o para obtener la información del calendario de las jornadas

Jornada (n) que se desea mostrar	Torneos que participan	Reparto de los enfrentamientos a mostrar entre los torneos
IMPAR	Si n = 1: **Torneo principal**	**Todos los enfrentamientos obtenidos pertenecen al mismo torneo: principal**
	Si n ≠ 1: **Torneo secundario**	**Todos los enfrentamientos obtenidos pertenecen al mismo torneo: secundario**
PAR	Para cualquier n participan ambos torneos: **Torneo principal** **Torneo secundario**	Los enfrentamientos obtenidos se dividen de la siguiente manera: **Torneo principal**: **Primera mitad del array** **Torneo secundario: Segunda mitad del array**

Mecanismo para la construcción de los enfrentamientos de la siguiente jornada al introducir los resultados de la que acaba de disputarse

Jornada (n) que acaba de disputarse	Torneos que tendrá la siguiente jornada (n + 1)	Reparto de los enfrentamientos a construir entre los torneos	
IMPAR	En la siguiente jornada (n+1) participarán ambos torneos: **Torneo principal** **Torneo secundario**	Si n=1: **Torneo principal**: ganadores de esta primera jornada **Torneo secundario**: perdedores de esta primera jornada	Una vez claramente repartidos entre ambos torneos, se guardan primero los duelos del principal y seguidamente los del secundario
		Si n = penúltima jornada: Se construye un único enfrentamiento entre el ganador de esta penúltima jornada (n) y el primer ganador de la antepenúltima (n-1)	
		Para el resto de jornadas: **Torneo principal:** ganadores de la primera mitad de los enfrentamientos de la jornada anterior (n-1) **Torneo secundario:** ganadores de la jornada que acaba de disputarse (n)	
PAR	En la siguiente jornada (n+1) siempre participará el **Torneo secundario**	Este **torneo secundario** estará formado por los perdedores de la primera mitad de los enfrentamientos de la jornada que acaba de disputarse (n) y por los ganadores de la segunda mitad de los enfrentamientos de la jornada que acaba de disputarse (n)	

Mecanismo para la sección de seguimiento del torneo: reparte a los participantes entre torneo principal, secundario o eliminados

Jornada (n) actual	Reparto entre los diferentes estados dentro de la aplicación
IMPAR	Si n = 1: En la primera jornada aún no se habrá disputado ningún enfrentamiento y por tanto todos los participantes se encuentran en el **torneo principal**
	En el resto de jornadas el reparto se hará de la siguiente manera: Pertenecerán al **torneo principal** los ganadores de la primera mitad de los enfrentamientos de la jornada anterior a la que se encuentran (n-1) En el **torneo secundario** se encuentran todos los participantes que deban de enfrentarse en el duelo de la jornada actual (n)
PAR	En el resto de jornadas el reparto se hará de la siguiente manera: Pertenecen al **torneo principal** todos los participantes que se encuentren en la primera mitad de los enfrentamientos de la jornada actual (n) Pertenecen al **torneo secundario** todos los participantes que se encuentren en la segunda mitad de los enfrentamientos de la jornada actual (n)

Una vez estén repartidos los participantes entre ambos torneos, se añaden a la lista de **eliminados** aquellos que participen en la competición y sin embargo no figuren en ninguno de los dos torneos

Figura 9.60. Interpretación de los enfrentamientos (torneo de tenis)

9.7. Componentes del dashboard

Todos los siguientes apartados contendrán primero su archivo TypeScript, seguido de su plantilla HTML.

9.7.1. Competiciones

Componente que muestra la lista de competiciones a las que se pertenece y la opción de crear los equipos en caso de ser profesor.

```
export class CompetitionsComponent implements OnInit {

  public competitions: Array<Competition>;

  constructor(
    public alertService: AlertService,
    public utilsService: UtilsService,
    public loadingService: LoadingService,
    public groupService: GroupService,
    public competitionService: CompetitionService,
    public teamService: TeamService) {
      this.utilsService.currentUser                                                    =
Login.toObject(localStorage.getItem(AppConfig.LS_USER));
      this.utilsService.role = Number(localStorage.getItem(AppConfig.LS ROLE));
      this.competitions = [];
    }

  ngOnInit() {
    if (this.utilsService.role === Role.TEACHER) {
      this.getTeacherCompetitions();
    } else if (this.utilsService.role === Role.STUDENT) {
      this.getStudentCompetitions();
    }
  }
  /**
   * This method returns the list of competitions from the
   * backend. This call is called by the teacher
   */
  getTeacherCompetitions(): void {
    this.groupService.getMyGroups().subscribe(
      ((groups: Array<Group>) =>
      groups.map( group => {
        this.competitionService.getMyCompetitionsByGroup(group).subscribe(
          ((competitions: Array<Competition>) =>
          competitions.map( competition => {
            this.loadingService.show();
            this.competitions.push(competition);
            this.loadingService.hide();
          })),
          ((error: Response) => {
            this.loadingService.hide();
            this.alertService.show(error.toString());
          }));
      })),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
  }
  /**
   * This method returns the list of competitions from the
   * backend. This call is called by the student
   */
  getStudentCompetitions(): void {
    this.competitionService.getMyCompetitions().subscribe(
      ((competitions: Array<Competition>) =>
```

```
      competitions.map( competition => {
        this.loadingService.show();
          this.competitions.push(competition);
        this.loadingService.hide();
      })),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));

    this.teamService.getTeamsStudent(+this.utilsService.currentUser.userId).subscribe(
      ((teams: Array<Team>) =>
      teams.map( team => {
          this.teamService.getMyCompetitionsGroup(+team.id).subscribe(
            ((competitions: Array<Competition>) =>
            competitions.map( competition => {
              this.loadingService.show();
                this.competitions.push(competition);
                this.loadingService.hide();
            })),
            ((error: Response) => {
              this.loadingService.hide();
              this.alertService.show(error.toString());
            }));
        this.loadingService.hide();
      })),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
  }

}
```

```
<div class="competitions-content" *ngIf="utilsService.role == 1 || utilsService.role ==
0">
  <mat-list>
    <h3 mat-subheader> {{ 'COMPETITIONS.TITLE' | translate }} </h3>

    <div *ngIf="utilsService.role == 1">
      <mat-list-item class="create-competition">
      <button        mat-raised-button        color="primary"        style="width:100%"
[matMenuTriggerFor]="menu">
        {{ 'COMPETITIONS.CREATE NEW COMPETITION' | translate }}
      </button>
      <mat-menu #menu="matMenu">
        <button mat-menu-item disabled><span><b>{{ 'COMPETITIONS.AVAILABLE' | translate
}}:</b></span></button>
        <a mat-menu-item routerLink="/competition/league/create">
          <mat-icon>chevron right</mat-icon>
          <span>{{ 'COMPETITIONS.LEAGUE' | translate }}</span>
        </a>
        <a mat-menu-item routerLink="/competition/tennis/create">
          <mat-icon>chevron_right</mat-icon>
          {{ 'TENNIS.TITLE' | translate }}
        </a>
      </mat-menu>
      </mat-list-item>
      <mat-divider></mat-divider>
      <mat-divider></mat-divider>
    </div>
    <mat-list-item>
      <h2 mat-dialog-title>{{ 'COMPETITIONS.SUBTITLE' | translate }}</h2>
   </mat-list-item>
  </mat-list>

  <mat-grid-list cols="1" rowHeight="9:1">
      <mat-grid-tile *ngIf="competitions.length === 0">
          <p *ngIf="utilsService.role == 1 ">{{ 'COMPETITIONS.NONE_TEACHER' | translate
}} </p>
          <p *ngIf="utilsService.role == 0 ">{{ 'COMPETITIONS.NONE STUDENT' | translate
}} </p>
      </mat-grid-tile>
    <div *ngIf="competitions">
      <mat-grid-tile *ngFor="let competition of competitions">
```

```
          <a mat-raised-button class="League" *ngIf="competition.type === 'Liga' "
[routerLink]="['/competition/league', competition.id]">
            <img src="../../../assets/img/liga-icon.svg" alt="angular">
            <span><strong>{{competition.name}}</strong>                              <br>
{{competition.grade.name}} {{competition.matter.name}}
              <strong>{{competition.mode}}</strong></span>
          </a>
          <a mat-raised-button class="Tennis" *ngIf="competition.type === 'Tenis' "
[routerLink]="['/competition/tennis', competition.id]">
            <img src="../../../assets/img/tenis-icon.svg" alt="angular">
            <span><strong>{{competition.name}}</strong> <br>{{competition.grade.name}}
{{competition.matter.name}}
              <strong>{{competition.mode}}</strong></span>
          </a>
      </mat-grid-tile>
    </div>
  </mat-grid-list>
  </div>
```

9.7.2. Crear competición de tipo liga

Componente que se encarga de la creación de la liga

```
export class CreateLeagueCompetitionComponent implements OnInit {

  public finished = false;
  public isLinear = true;

  public competitionFormGroup: FormGroup;
  public journeysFormGroup: FormGroup;
  public participantsFormGroup: FormGroup;
  public informationFormGroup: FormGroup;
  public groups = [];

  public participant: any;
  public participants = new Array<any>();
  // forms
  public newCompetition: Competition;
  public selectedParticipants: Array<number>;
  public newJourneys: Array<any>;
  public newInformation: string;
  public newCompetitionPost: Competition;
  public journeys = new Array();
  public match: any;

  constructor( public alertService: AlertService,
    public utilsService: UtilsService,
    public loadingService: LoadingService,
    public groupService: GroupService,
    public competitionService: CompetitionService,
    public journeyService: JourneyService,
    public matchesService: MatchesService,
    public teamService: TeamService,
    private _formBuilder: FormBuilder) {
      this.utilsService.currentUser =
Login.toObject(localStorage.getItem(AppConfig.LS USER));
      this.utilsService.role = Number(localStorage.getItem(AppConfig.LS ROLE));
   }

  ngOnInit() {

    if (this.utilsService.role === Role.TEACHER) {

      // Defining 3 forms:
    this.competitionFormGroup = this._formBuilder.group({
      name: ['', Validators.required],
      groupId: ['', Validators.required],
      mode: ['', Validators.required],
      numJourneys: ['', Validators.required]
    });

    this.participantsFormGroup = this. formBuilder.group({
```

```
      participantId: ['']
    });

    this.journeysFormGroup = this._formBuilder.group({
      journeys: this._formBuilder.array([
        this. formBuilder.group({
          date: ['']
        })
      ])
    });

    this.informationFormGroup = this. formBuilder.group({
      information: ['']
    });

    // Getting teacher's group
      this.loadingService.show();
      this.groupService.getMyGroups().subscribe(
        ((groups: Array<Group>) => {
          this.groups = groups;
          this.loadingService.hide();
        }),
        ((error: Response) => {
          this.loadingService.hide();
          this.alertService.show(error.toString());
        }));
    }
  }
  /**
   * This method saves the competition in a variable
   * and calls the method to get participants
   */
  competitionStep(value: Competition) {
    this.loadingService.show();
    this.newCompetition = value;
    this.newCompetition.type = 'Liga';
    this.getParticipants(); // getting participants for the next step
  }
  /**
   * This method gets the participants of the group
   * for the competition
   */
  getParticipants(): void {
    if ( this.newCompetition.mode === 'Individual' ) {
      this.groupService.getMyGroupStudents(this.newCompetition.groupId).subscribe(
      ( (students: Array<Student>) => {
        for (let  n = 0;  n < students.length;  n++) {
         this.participant = {
          id: students[_n].id,
          name:  students[ n].name,
          surname: students[ n].surname,
          selected: false
          };
          this.participants.push(this.participant);
        }
        this.loadingService.hide();
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
      } else {
        this.groupService.getGroupTeams(this.newCompetition.groupId).subscribe(
        ( (teams: Array<Team>) => {
          for (let _a = 0; _a < teams.length; _a++) {
              this.participant = {
                id: teams[_a].id,
                name: teams[ a].name,
                surname: '',
                selected: false
                };
             this.participants.push(this.participant);
          }
          this.loadingService.hide();
        }),
        ((error: Response) => {
          this.loadingService.hide();
```

```
          this.alertService.show(error.toString());
        }));
      }
  }
  /**
   * This method saves the participants in a variable
   * and prepares the journeys for the next step
   */
  participantStep(list) {
    this.loadingService.show();
    this.selectedParticipants = list.selectedOptions.selected.map(item => item.value);
    // Add the journeys to the next step
    for (let  n = 0;  n < this.newCompetition.numJourneys - 1;  n++) {
      let journeys = <FormArray>this.journeysFormGroup.get('journeys');
      journeys.push(this._formBuilder.group({
       date: ['']
      }));
    }
    this.loadingService.hide();
  }
  /** This method saves the journeys in a variable */
  journeyStep(value) {
    this.loadingService.show();
    this.newJourneys = value.journeys;
    this.loadingService.hide();
  }
  /**
   * This method saves the information in a variable and
   * calls the method to subit the competition
   */
  informationStep(value: Competition) {
    this.loadingService.show();
    this.newInformation = value.information;
    this.onSubmitCompetition();
  }
  /**
   * This method posts the competition
   * and calls the method to submit the journeys
   */
  onSubmitCompetition(): void {
    this.newCompetition.information = this.newInformation;
    this.competitionService.postCompetition(this.newCompetition)
    .subscribe( (competition => {
      this.newCompetitionPost = competition;
      this.onSubmitJourneys();
    }),
    ((error: Response) => {
      this.loadingService.hide();
      this.alertService.show(error.toString());
    }));
  }
  /**
   * This method posts the journeys
   * and calls the method to submit the relations
   */
  onSubmitJourneys(): void {
    // Adding to the journeys: number and competitionId
    for ( let  n = 0;  n < this.newCompetition.numJourneys;  n++) {
      this.newJourneys[_n].number = _n + 1;
      this.newJourneys[_n].competitionId = +this.newCompetitionPost.id;
      if ( this.newJourneys[_n].date === '' ) { this.newJourneys[_n].date = null; }
    }
    // POST JOURNEYS
    for (let _a = 0; _a < this.newJourneys.length; _a++) {
      this.journeyService.postJourney(this.newJourneys[_a])
      .subscribe( (journey => {
       this.journeys.push(journey);
       if (this.journeys.length === this.newJourneys.length) {
         this.journeys.sort(function (a, b) {
          return (a.number - b.number);
         });
        this.onSubmitRelations();
       }
      }),
       ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
```

```
      }));
    }
  }
  /**
   * This method posts the relations beetween competition and participants
   * and calls the method to submit the journeys
   */
  onSubmitRelations(): void {
    let count = 0;
        if ( this.newCompetition.mode === 'Individual' ) {
          for ( let _i = 0; _i < this.selectedParticipants.length; _i++ ) {
            this.competitionService.relCompetitionStudent(this.newCompetitionPost.id,
this.selectedParticipants[ i]).subscribe(
              ( res => {
                count++;
                if ( count === this.selectedParticipants.length ) { this.league(); }
              }),
              ((error: Response) => {
                this.loadingService.hide();
                this.alertService.show(error.toString());
              }));
          }
        } else {
          for ( let  i = 0;  i < this.selectedParticipants.length;  i++ ) {
            this.teamService.relCompetitionTeam(this.newCompetitionPost.id,
this.selectedParticipants[_i]).subscribe(
              ( res => {
                count++;
                if ( count === this.selectedParticipants.length ) { this.league(); }
              }),
              ((error: Response) => {
                this.loadingService.hide();
                this.alertService.show(error.toString());
              }));
          }
        }
  }
  /**
   * This method prepares the matches of the competition
   * and posts the matches
   */
  league(): void {
    // adding ghost participant
    if (this.selectedParticipants.length % 2 !== 0) {
      this.selectedParticipants.push(0);
    }
    this.selectedParticipants = this.selectedParticipants.sort(function() {return
Math.random() - 0.5});
  // Matches
  let count2 = 0;
   for (let  j = 0;  j < this.journeys.length;  j++) {
    for (let  m = 0;  m < (this.selectedParticipants.length / 2);  m++) {
      this.match = {
        playerOne : +this.selectedParticipants[_m],
        playerTwo : +this.selectedParticipants[this.selectedParticipants.length - 1 -
 m],
        journeyId : +this.journeys[ j].id
      };
        // POST MATCHES
        this.matchesService.postMatch(this.match)
        .subscribe( (match => {
          count2++;
          if ( count2 === (this.journeys.length * (this.selectedParticipants.length /
2)) ) {
            this.finished = true;
            this.loadingService.hide();
          }
        }),
        ((error: Response) => {
          this.loadingService.hide();
          this.alertService.show(error.toString());
        }));
    }
    // Changing position on selectedParticipants
    let studentBefore =  this.selectedParticipants[1];
    for (let _s = 1; _s < (this.selectedParticipants.length - 1); _s++) {
      this.selectedParticipants[_s] = this.selectedParticipants[_s + 1];
```

```
    }
    this.selectedParticipants[this.selectedParticipants.length - 1] = studentBefore;
   }
}
}
```

```
<div class="stepper-content">
<h4>{{ 'COMPETITION_CREATION.TITLE' | translate }} {{ 'COMPETITIONS.LEAGUE' |
translate }}:</h4>
<mat-horizontal-stepper [linear]="isLinear" #stepper="matHorizontalStepper">
  <mat-step label="{{ 'COMPETITIONS.INITIALIZATION' | translate }}"
[stepControl]="competitionFormGroup" editable="false">

    <form [formGroup]="competitionFormGroup"
(ngSubmit)="competitionStep(competitionFormGroup.value)">
       <div class="form-group">
         <mat-form-field>
           <input matInput type="text" class="form-control" placeholder="{{
'COMPETITIONS.NAME' | translate }}" required formControlName="name">
         </mat-form-field>
       </div>

       <div class="form-group">
         <mat-form-field>
           <mat-select class="form-control" placeholder="{{ 'GROUPS.SINGULAR' |
translate }}" formControlName="groupId" required>
             <mat-option *ngFor="let group of groups" [value]="group.id">
              {{ group.grade.name + ' ' + group.matter.name}}
             </mat-option>
           </mat-select>
         </mat-form-field>
        </div>

        <div class="form-group-radio">
           <mat-radio-group class="form-control-radio" required
formControlName="mode">
             <mat-radio-button [value]="'Individual'">
                {{ 'COMPETITIONS.INDIVIDUAL' | translate }}
             </mat-radio-button>
             <mat-radio-button [value]="'Equipos'">
                {{ 'COMPETITIONS.TEAMS' | translate }}
            </mat-radio-button>
           </mat-radio-group>
        </div>

        <div class="form-group">
           <mat-form-field>
             <input matInput type="number" class="form-control" required
formControlName="numJourneys" placeholder="{{ 'COMPETITIONS.NUMBER JOURNEYS' |
translate }}">
           </mat-form-field>
         </div>
       <div>
        <button mat-raised-button color="primary" matStepperNext
[disabled]="!competitionFormGroup.valid">{{ 'COMPETITIONS.NEXT' | translate
}}</button>
      </div>
   </form>
  </mat-step>


  <mat-step label="Participantes" [stepControl]="participantsFormGroup"
editable="false">
      <p>{{ 'COMPETITION CREATION.PARTICIPANT' | translate }}</p>
      <p>{{ 'COMPETITION CREATION.MIN 2' | translate }}</p>
      <mat-selection-list #list>
          <mat-list-option *ngFor="let participant of participants"
           [selected]="participant.selected" [value]="participant.id">
              {{participant.name}} {{participant.surname}}
          </mat-list-option>
      </mat-selection-list>
       <div>
       <button mat-raised-button color="primary" matStepperNext
(click)="participantStep(list)"
       [disabled]="list.selectedOptions.selected.length < 2">{{ 'COMPETITIONS.NEXT' |
translate }}</button>
```

```
      </div>
  </mat-step>


  <mat-step label="{{ 'COMPETITIONS.JOURNEYS' | translate }}" class="mat-horizontal-
step" [stepControl]="journeysFormGroup" editable="false">

    <form [formGroup]="journeysFormGroup"
(ngSubmit)="journeyStep(journeysFormGroup.value)">
        <p>{{ 'COMPETITIONS.INTR_JOURNEY' | translate }}</p>

       <div class="form-group" formArrayName="journeys">
          <div *ngFor="let journey of journeysFormGroup.get('journeys')['controls']
;let i=index;">

                <div class="panel-body" [formGroupName]="i">
                    <div class="form-group">
                      <mat-form-field>
                          <input matInput [matDatepicker]="picker"
formControlName="date"
                          placeholder=" {{ 'COMPETITIONS.JOURNEY' | translate }} {{i +
1}}">
                          <mat-datepicker-toggle matSuffix [for]="picker"></mat-
datepicker-toggle>
                          <mat-datepicker #picker></mat-datepicker>
                      </mat-form-field>
                    </div>
                </div>
          </div>
       </div>
       <div>
          <button mat-raised-button color="primary" matStepperNext
[disabled]="!journeysFormGroup.valid">{{ 'COMPETITIONS.NEXT' | translate }}</button>
       </div>
    </form>
   </mat-step>

  <mat-step label="{{ 'COMPETITIONS.MORE_INFO' | translate }}" class="mat-horizontal-
step" [stepControl]="informationFormGroup"
  (click)="journeyStep(journeysFormGroup.value)" editable="false">
   <form [formGroup]="informationFormGroup"
(ngSubmit)="informationStep(informationFormGroup.value)">
      <p>{{ 'COMPETITIONS.EXPL_INFO' | translate }}</p>
        <div>
       <mat-form-field class="text-info">
           <textarea matInput placeholder="Deje su comentario"
formControlName="information"></textarea>
        </mat-form-field>
      </div>

     <div>
        <button mat-raised-button color="primary" matStepperNext
[disabled]="!informationFormGroup.valid">{{ 'COMPETITIONS.NEXT' | translate
}}</button>
     </div>
    </form>
  </mat-step>

  <mat-step label="{{ 'COMPETITIONS.FINISHED2' | translate }}"
(click)="informationStep(informationFormGroup.value)">
     <div *ngIf="finished">
      <p>{{ 'COMPETITIONS.FINISHED' | translate }}</p>
      <div class="final-buttons">
        <a mat-raised-button color="primary" class="Clasificacion"
routerLink="/competitions">
          <strong>{{ 'COMPETITIONS.DONE' | translate }}</strong>
        </a>
      </div>
    </div>
  </mat-step>
 </mat-horizontal-stepper>
</div>
```

9.7.3. Crear competición de tipo torneo de tenis

Componente que se encarga de la creación del torneo de tenis

```
export class CreateTennisCompetitionComponent implements OnInit {

  public finished = false;
  public isLinear = true;

  public competitionFormGroup: FormGroup;
  public journeysFormGroup: FormGroup;
  public participantsFormGroup: FormGroup;
  public informationFormGroup: FormGroup;
  public groups = [];

  public participant: any;
  public participants = new Array<any>();
  public exp: number;
  // forms
  public newCompetition: Competition;
  public selectedParticipants: Array<number>;
  public newJourneys: Array<any>;
  public newInformation: string;
  public newCompetitionPost: Competition;
  public journeys = new Array();
  public match: any;

  constructor(
    public alertService: AlertService,
    public utilsService: UtilsService,
    public loadingService: LoadingService,
    public groupService: GroupService,
    public competitionService: CompetitionService,
    public journeyService: JourneyService,
    public matchesService: MatchesService,
    public teamService: TeamService,
    private  formBuilder: FormBuilder) {
      this.utilsService.currentUser                                                    =
Login.toObject(localStorage.getItem(AppConfig.LS_USER));
      this.utilsService.role = Number(localStorage.getItem(AppConfig.LS_ROLE));
    }

  ngOnInit() {
    if (this.utilsService.role === Role.TEACHER) {

      // Defining 3 forms:
    this.competitionFormGroup = this._formBuilder.group({
      name: ['', Validators.required],
      groupId: ['', Validators.required],
      mode: ['', Validators.required]
    });

    this.participantsFormGroup = this. formBuilder.group({
      participantId: ['']
    });

    this.journeysFormGroup = this. formBuilder.group({
      journeys: this. formBuilder.array([
        this. formBuilder.group({
          date: ['']
        })
      ])
    });

    this.informationFormGroup = this. formBuilder.group({
      information: ['']
    });

    // Getting teacher's group
      this.loadingService.show();
      this.groupService.getMyGroups().subscribe(
        ((groups: Array<Group>) => {
          this.groups = groups;
          this.loadingService.hide();
```

```
        }),
        ((error: Response) => {
          this.loadingService.hide();
          this.alertService.show(error.toString());
        }));
    }
  }
  /**
   * This method saves the competition in a variable
   * and calls the method to get participants
   */
  competitionStep(value: Competition) {
    this.loadingService.show();
    this.newCompetition = value;
    this.newCompetition.type = 'Tenis';
    this.getParticipants(); // getting participants for the next step
  }
  /**
   * This method gets the participants of the group
   * for the competition
   */
  getParticipants(): void {
    if ( this.newCompetition.mode === 'Individual' ) {
      this.groupService.getMyGroupStudents(this.newCompetition.groupId).subscribe(
      ( (students: Array<Student>) => {
         for (let _n = 0; _n < students.length; _n++) {
         this.participant = {
          id: students[ n].id,
          name:  students[ n].name,
          surname: students[_n].surname,
          selected: false
          };
          this.participants.push(this.participant);
        }
        this.loadingService.hide();
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
      } else {
        this.groupService.getGroupTeams(this.newCompetition.groupId).subscribe(
        ( (teams: Array<Team>) => {
          for (let _a = 0; _a < teams.length; _a++) {
              this.participant = {
                id: teams[ a].id,
                name: teams[ a].name,
                surname: '',
                selected: false
                };
             this.participants.push(this.participant);
          }
          this.loadingService.hide();
        }),
        ((error: Response) => {
          this.loadingService.hide();
          this.alertService.show(error.toString());
        }));
      }
  }
  /**
   * This method saves the participants in a variable
   * and prepares the journeys for the next step
   */
  participantStep(list) {
    this.loadingService.show();
    this.selectedParticipants = list.selectedOptions.selected.map(item => item.value);
    // Computing number of journeys and participants (Math.pow(2, exp))
      for (let  p = 1; !this.exp;  p++) {
        if ( Math.pow(2, _p) >= this.selectedParticipants.length ) {
        this.exp = _p;
        this.newCompetition.numJourneys = this.exp * 2;
        }
      }
    // Add the journeys to the next step
    for (let _n = 0; _n < this.newCompetition.numJourneys - 1; _n++) {
      let journeys = <FormArray>this.journeysFormGroup.get('journeys');
```

```
      journeys.push(this. formBuilder.group({
       date: ['']
      }));
    }
    this.loadingService.hide();
  }
  /** This method saves the journeys in a variable */
  journeyStep(value) {
    this.loadingService.show();
    this.newJourneys = value.journeys;
    this.loadingService.hide();
  }
  /**
   * This method saves the information in a variable and
   * calls the method to subit the competition
   */
  informationStep(value: Competition) {
    this.loadingService.show();
    this.newInformation = value.information;
    this.onSubmitCompetition();
  }
  /**
   * This method posts the competition
   * and calls the method to submit the journeys
   */
  onSubmitCompetition(): void {
    this.newCompetition.information = this.newInformation;
    this.competitionService.postCompetition(this.newCompetition)
    .subscribe( (competition => {
      this.newCompetitionPost = competition;
      this.onSubmitJourneys();
    }),
    ((error: Response) => {
      this.loadingService.hide();
      this.alertService.show(error.toString());
    }));
  }
 /**
   * This method posts the journeys
   * and calls the method to submit the relations
   */
  onSubmitJourneys(): void {
      // Adding to the journeys: number and competitionId
    for ( let _n = 0; _n < this.newCompetition.numJourneys; _n++) {
      this.newJourneys[ n].number =  n + 1;
      this.newJourneys[ n].competitionId = +this.newCompetitionPost.id;
      if ( this.newJourneys[ n].date === '' ) {
       this.newJourneys[_n].date = null;
      }
    }
    // POST JOURNEYS
    for (let  a = 0;  a < this.newJourneys.length;  a++) {
      this.journeyService.postJourney(this.newJourneys[_a])
        .subscribe( (journey => {
         this.journeys.push(journey);
         if (this.journeys.length === this.newJourneys.length) {
           this.journeys.sort(function (a, b) {
            return (a.number - b.number);
           });
          this.onSubmitRelations();
         }
        }),
         ((error: Response) => {
          this.loadingService.hide();
          this.alertService.show(error.toString());
        }));
    }
  }
  /**
   * This method posts the relations beetween competition and participants
   * and calls the method to submit the journeys
   */
  onSubmitRelations(): void {
    let countParticipant = 0;
    if ( this.newCompetition.mode === 'Individual' ) {
      for ( let _i = 0; _i < this.selectedParticipants.length; _i++ ) {
```

```
        this.competitionService.relCompetitionStudent(this.newCompetitionPost.id,
this.selectedParticipants[_i]).subscribe(
        ( res => {
        countParticipant++;
        if ( countParticipant === this.selectedParticipants.length ) {
this.putFirstMatches(); }
        }),
        ((error: Response) => {
        this.loadingService.hide();
         this.alertService.show(error.toString());
        }));
      }
    } else {
      for ( let _i = 0; _i < this.selectedParticipants.length; _i++ ) {
        this.teamService.relCompetitionTeam(this.newCompetitionPost.id,
this.selectedParticipants[_i]).subscribe(
         ( res => {
         countParticipant++;
         if ( countParticipant === this.selectedParticipants.length ) {
this.putFirstMatches(); }
         }),
        ((error: Response) => {
          this.loadingService.hide();
          this.alertService.show(error.toString());
         }));
      }
    }
  }
  /**
   * This method prepares the first matches of the competition
   * and posts the matches
   */
  putFirstMatches(): void {
    // adding ghost participant
    if (this.selectedParticipants.length !== Math.pow(2, this.exp)) {
      for (let _s = 1;  Math.pow(2, this.exp) > this.selectedParticipants.length;
_s++) {
        this.selectedParticipants.push(0);
      }
    }
    this.selectedParticipants = this.selectedParticipants.sort(function() {return
Math.random() - 0.5});
    let countMatches = 0;
      for (let _m = 0; _m < (this.selectedParticipants.length / 2); _m++) {
        this.match = {
          playerOne : +this.selectedParticipants[ m],
          playerTwo : +this.selectedParticipants[this.selectedParticipants.length - 1
- _m],
          journeyId : +this.journeys[0].id
        };
          // POST MATCHES
          this.matchesService.postMatch(this.match)
          .subscribe( (match => {
            countMatches++;
            if ( countMatches === (this.selectedParticipants.length / 2)) {
              this.finished = true;
              this.loadingService.hide();
              this.alertService.show('La competición se ha creado correctamente');
            }
          }),
          ((error: Response) => {
            this.loadingService.hide();
            this.alertService.show(error.toString());
          }));
      }
    }
}
```

```
<div class="stepper-content">
    <h4>{{ 'COMPETITION CREATION.TITLE' | translate }} {{ 'COMPETITIONS.TENNIS' |
translate }}:</h4>
    <mat-horizontal-stepper [linear]="isLinear" #stepper="matHorizontalStepper">
      <mat-step label="{{ 'COMPETITIONS.INITIALIZATION' | translate }}"
[stepControl]="competitionFormGroup" editable="false">
```

```
        <form [formGroup]="competitionFormGroup"
(ngSubmit)="competitionStep(competitionFormGroup.value)">
           <div class="form-group">
             <mat-form-field>
               <input matInput type="text" class="form-control" placeholder="{{
'COMPETITIONS.NAME' | translate }}" required formControlName="name">
             </mat-form-field>
           </div>

           <div class="form-group">
             <mat-form-field>
               <mat-select class="form-control" placeholder="{{ 'GROUPS.SINGULAR' |
translate }}" formControlName="groupId" required>
                 <mat-option *ngFor="let group of groups" [value]="group.id">
                  {{ group.grade.name + ' ' + group.matter.name}}
                 </mat-option>
               </mat-select>
             </mat-form-field>
            </div>

            <div class="form-group-radio">
               <mat-radio-group class="form-control-radio" required
formControlName="mode">
                 <mat-radio-button [value]="'Individual'">
                  {{ 'COMPETITIONS.INDIVIDUAL' | translate }}
                 </mat-radio-button>
                 <mat-radio-button [value]="'Equipos'">
                  {{ 'COMPETITIONS.TEAMS' | translate }}
                </mat-radio-button>
               </mat-radio-group>
            </div>
           <div>
            <button mat-button matStepperNext
[disabled]="!competitionFormGroup.valid">{{ 'COMPETITIONS.NEXT' | translate
}}</button>
          </div>
       </form>
      </mat-step>


      <mat-step label="Participantes" [stepControl]="participantsFormGroup"
editable="false">
          <p>{{ 'COMPETITION_CREATION.PARTICIPANT' | translate }}</p>
          <p>{{ 'COMPETITION_CREATION.MIN_2' | translate }}</p>
          <mat-selection-list #list>
              <mat-list-option *ngFor="let participant of participants"
               [selected]="participant.selected" [value]="participant.id">
                  {{participant.name}} {{participant.surname}}
              </mat-list-option>
          </mat-selection-list>
           <div>
           <button mat-button matStepperNext (click)="participantStep(list)"
           [disabled]="list.selectedOptions.selected.length < 2">{{
'COMPETITIONS.NEXT' | translate }}</button>
          </div>
      </mat-step>


      <mat-step label="{{ 'COMPETITIONS.JOURNEYS' | translate }}" class="mat-
horizontal-step" [stepControl]="journeysFormGroup" editable="false">

        <form [formGroup]="journeysFormGroup"
(ngSubmit)="journeyStep(journeysFormGroup.value)">
            <p>{{ 'COMPETITIONS.INTR_JOURNEY' | translate }}</p>

           <div class="form-group" formArrayName="journeys">
              <div *ngFor="let journey of
journeysFormGroup.get('journeys')['controls'] ;let i=index;">

                    <div class="panel-body" [formGroupName]="i">
                        <div class="form-group">
                          <mat-form-field>
                              <input matInput [matDatepicker]="picker"
formControlName="date"
                              placeholder=" {{ 'COMPETITIONS.JOURNEY' | translate }}
{{i + 1}}">
```

```
                              <mat-datepicker-toggle matSuffix [for]="picker"></mat-
datepicker-toggle>
                              <mat-datepicker #picker></mat-datepicker>
                            </mat-form-field>
                          </div>
                      </div>
                  </div>
              </div>
              <div>
                 <button mat-button matStepperNext
[disabled]="!journeysFormGroup.valid">{{ 'COMPETITIONS.NEXT' | translate }}</button>
              </div>
          </form>

        </mat-step>


       <mat-step label="{{ 'COMPETITIONS.MORE INFO' | translate }}" class="mat-
horizontal-step" [stepControl]="informationFormGroup"
       (click)="journeyStep(journeysFormGroup.value)" editable="false">
        <form [formGroup]="informationFormGroup"
(ngSubmit)="informationStep(informationFormGroup.value)">
           <p>{{ 'COMPETITIONS.EXPL INFO' | translate }}</p>
             <div>
            <mat-form-field class="text-info">
                <textarea matInput placeholder="Deje su comentario"
formControlName="information"></textarea>
             </mat-form-field>
           </div>

          <div>
             <button mat-button matStepperNext
[disabled]="!informationFormGroup.valid">{{ 'COMPETITIONS.NEXT' | translate
}}</button>
          </div>
         </form>
       </mat-step>

       <mat-step     label="{{     'COMPETITIONS.FINISHED2'     |     translate     }}"
(click)="informationStep(informationFormGroup.value)">
          <div *ngIf="finished">
           <p>{{ 'COMPETITIONS.FINISHED' | translate }}</p>
           <div class="final-buttons">
             <a mat-button class="Clasificacion" routerLink="/competitions">
               <strong>{{ 'COMPETITIONS.DONE' | translate }}</strong>
             </a>
           </div>
         </div>
       </mat-step>
      </mat-horizontal-stepper>
     </div>
```

9.7.4. Crear equipos

Componente que se encarga de la creación de los equipos

```
export class CreateTeamsComponent implements OnInit {

  public firstFormGroup: FormGroup;
  public secondFormGroup: FormGroup;
  public show = 1;
  public groups = [];
  public team: any;
  public teams = new Array();

  public students = new Array<Student>();
  public numStudents: number;

  constructor( public alertService: AlertService,
    public utilsService: UtilsService,
    public loadingService: LoadingService,
    public groupService: GroupService,
```

```
    public competitionService: CompetitionService,
    public teamService: TeamService,
    private _formBuilder: FormBuilder) {
      this.utilsService.currentUser =
Login.toObject(localStorage.getItem(AppConfig.LS_USER));
      this.utilsService.role = Number(localStorage.getItem(AppConfig.LS ROLE));
     }

  ngOnInit() {

    if (this.utilsService.role === Role.TEACHER) {
    this.firstFormGroup = this. formBuilder.group({
      groupId: ['', Validators.required],
      teams: this._formBuilder.array([
        this._formBuilder.group({
          name: ['', Validators.required]
        })
      ])
    });
    this.secondFormGroup = this._formBuilder.group({
      teamsId: this._formBuilder.array([
        this. formBuilder.group({
          teamId: ['', Validators.required]
        })
      ])
    });

    let teams = <FormArray>this.firstFormGroup.get('teams');
    teams.push(this. formBuilder.group({
      name: ['', Validators.required]
    }));

    // Getting teacher's group
      this.loadingService.show();
      this.groupService.getMyGroups().subscribe(
        ((groups: Array<Group>) => {
          this.groups = groups;
          this.loadingService.hide();
        }),
        ((error: Response) => {
          this.loadingService.hide();
          this.alertService.show(error.toString());
        }));
    }
  }
    /**
   * This method submits the list of teams with the group
   * and gets the students for the next step
   */
  onSubmit(value) {
    this.loadingService.show();
        for (let  m = 0;  m < (value.teams.length);  m++) {
         this.team = {
          name: value.teams[_m].name,
          groupId: value.groupId
          };
          this.teamService.postTeam(this.team)
          .subscribe( (team => {
           this.teams.push(team);
           if ( this.teams.length === value.teams.length - 1 ) {
            this.show = this.show + 1;
           }
           this.loadingService.hide();
            }),
           ((error: Response) => {
            this.loadingService.hide();
            this.alertService.show(error.toString());
            }));
        }
      // GET STUDENTS of the group
      this.groupService.getMyGroupStudents(value.groupId).subscribe(
      ( (students: Array<Student>) => {
        this.students = students;
        this.numStudents = this.students.length;
        for (let _n = 0; _n < this.numStudents - 1; _n++) {
          let teamsId = <FormArray>this.secondFormGroup.get('teamsId');
          teamsId.push(this._formBuilder.group({
```

```
            teamId: ['', Validators.required]
          }));
        }
        this.show = this.show + 1;
        this.loadingService.hide();
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));

  }
    /**
   * This method submits the relation of each student with a team
   */
  onSubmit2(value) {
    this.loadingService.show();
      for ( let  s = 0;  s < this.numStudents;  s++ ) {
      this.teamService.relTeamStudent(value.teamsId[ s].teamId,
this.students[_s].id).subscribe(
        ( res => {
          if (  s === this.numStudents - 1 ) { this.show = this.show + 1; }
          this.loadingService.hide();
        }),
        ((error: Response) => {
          this.loadingService.hide();
          this.alertService.show(error.toString());
        }));
      }
  }
      /**
   * Remove a team
   */
  removeTeam(i) {
    let teams = <FormArray>this.firstFormGroup.get('teams');
    teams.removeAt(i);
  }
    /**
   * Add a team
   */
  addTeam() {
    let teams = <FormArray>this.firstFormGroup.get('teams');
    teams.push(this._formBuilder.group({
      name: ['', Validators.required]
    }));
  }
  /**
   * This method returns the list of students of the group.
   * This call is called by the template of create-teams.
   */
  getValue(i: number) {
    return (this.students[i].name);
  }
}
```

```
<div *ngIf="utilsService.role == 1" class="creation-teams-content">
  <div class="first" *ngIf="show === 1">
    <h3 mat-subheader>{{ 'TEAMS.TITLE' | translate }}</h3>

  <form [formGroup]="firstFormGroup" (ngSubmit)="onSubmit(firstFormGroup.value)">
      <p>{{ 'TEAMS.GROUP' | translate }}:</p>
      <mat-card>
        <div class="form-group">
          <mat-form-field>
            <mat-select class="form-control" placeholder="{{ 'GROUPS.SINGULAR' |
translate }}" formControlName="groupId" required>
              <mat-option *ngFor="let group of groups" [value]="group.id">
                {{ group.grade.name + ' ' + group.matter.name}}
              </mat-option>
            </mat-select>
          </mat-form-field>
          </div>
      </mat-card>

      <p>{{ 'TEAMS.INTRO' | translate }}</p>
    <mat-card>
```

```
      <div class="form-group" formArrayName="teams">
          <div *ngFor="let team of firstFormGroup.controls.teams.controls ;let
i=index;">
            <mat-card-header>
              <div>
                {{ 'TEAMS.TEAM' | translate }} {{i + 1}}
                <span *ngIf="firstFormGroup.controls.teams.controls.length > 2"
(click)="removeTeam(i)">
                    <button mat-button color="warn">{{ 'TEAMS.DELETE' | translate }}
{{i + 1}}</button>
                </span>
              </div>
            </mat-card-header>
              <mat-card-content>
                <div class="panel-body" [formGroupName]="i">
                  <div class="form-group">
                    <mat-form-field>
                      <input matInput type="text" class="form-control" placeholder="{{
'TEAMS.TEAM' | translate }} {{i + 1}}" required formControlName="name">
                    </mat-form-field>
                  </div>
                </div>
              </mat-card-content>
          </div>
      </div>
   </mat-card>

   <div>
      <a mat-raised-button color="accent" (click)="addTeam()" class="btn-team">
          {{ 'TEAMS.MORE' | translate }} <mat-icon mat-list-icon>group add</mat-
icon></a>
  </div>
  <div class="sbt-btn">
      <button mat-raised-button color="primary" (click)="onSubmit"
      [disabled]="!firstFormGroup.valid">{{ 'COMPETITIONS.NEXT' | translate
}}</button>
  </div>
  </form>

 </div>

 <div *ngIf="show === 3">
  <form [formGroup]="secondFormGroup" (ngSubmit)="onSubmit2(secondFormGroup.value)">
   <p>{{ 'TEAMS.STUDENT' | translate }}:</p>
   <mat-card>
    <div class="form-group" formArrayName="teamsId">
      <div *ngFor="let teamId of secondFormGroup.controls.teamsId.controls ;let
i=index;">
        <div class="panel-body" [formGroupName]="i">
          <mat-form-field class="mat-form-student">
            <mat-select placeholder="Equipo de {{ getValue(i) }}"
formControlName="teamId">
              <mat-option *ngFor="let team of teams" [value]="team.id">
                {{team.name}}
              </mat-option>
            </mat-select>
          </mat-form-field>
        </div>
      </div>
    </div>
    <div class="sbt-btn">
        <button mat-raised-button color="primary" (click)="onSubmit2"
        [disabled]="!secondFormGroup.valid">{{ 'COMPETITIONS.NEXT' | translate
}}</button>
    </div>
   </mat-card>
  </form>
 </div>

 <div *ngIf="show === 4">

    <h3>{{ 'TEAMS.FINISHED' | translate }}.</h3>

      <div class="final-button">
          <a mat-button [routerLink]="'/competitions'">
            <mat-icon mat-list-icon>navigate_before</mat-icon>
            {{ 'TEAMS.RETURN' | translate }}
```

```
          </a>
      </div>
 </div>

</div>
<div *ngIf="utilsService.role === 0 || utilsService.role == 2" class="creation-teams-
content">
  {{ 'TEAMS.TEACHER' | translate }}.
</div>
```

9.7.5. Eliminar competición

Componente que se encarga de la eliminación de una competición determinada

```
export class DeleteCompetitionComponent implements OnInit {

  public competition: Competition;
  public journeys: Journey[];

  constructor(
    public alertService: AlertService,
    public loadingService: LoadingService,
    public competitionService: CompetitionService,
    private route: ActivatedRoute,
    private router: Router,
    private dialogRef: MatDialogRef<DeleteCompetitionComponent>,
    @Inject(MAT_DIALOG_DATA) public data: any) { }

  ngOnInit() {
    this.competition = this.data.competition;
    this.journeys = this.data.journeys;
  }

  cancel(): void {
    this.dialogRef.close();
  }

  delete(): void {
    this.loadingService.show();
    this.deleteParticipants();
  }

  deleteParticipants() {
    this.competitionService.deleteParticipantsCompetition(this.competition)
    .subscribe( (response => {
      this.deleteMatches();
    }),
    ((error: Response) => {
      this.loadingService.hide();
      this.alertService.show(error.toString());
    }));
  }

  deleteMatches() {
    let journeysDeleted = 0;
    for (let  j = 0;  j < this.journeys.length;  j++) {
      this.competitionService.deleteMatchesCompetition(+this.journeys[ j].id)
      .subscribe( (response => {
        journeysDeleted = journeysDeleted + 1;
        if (journeysDeleted === this.journeys.length) {
          this.deleteJourneys();
        }
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
    }

  }

  deleteJourneys() {
```

```
    this.competitionService.deleteJourneysCompetition(+this.competition.id)
    .subscribe( (response => {
      this.deleteCompetition();
    }),
    ((error: Response) => {
      this.loadingService.hide();
      this.alertService.show(error.toString());
    }));
  }

  deleteCompetition() {
    this.competitionService.deleteCompetition(+this.competition.id)
    .subscribe( (response => {
      this.loadingService.hide();
      this.alertService.show('La competición ha sido eliminada');
      this.dialogRef.close();
      this.router.navigate(['/competitions']);
    }),
    ((error: Response) => {
      this.loadingService.hide();
      this.alertService.show(error.toString());
    }));
  }

}
```

```
<div class="delete-content">
  <p><b>{{ 'COMPETITIONS.DELETE QUESTION' | translate }}</b></p>
  <button mat-raised-button class="delete" (click)="delete()">{{
'COMPETITIONS.DELETE2' | translate }}</button>
  <button mat-button class="cancel" (click)="cancel()"> <b>{{ 'COMPETITIONS.CANCEL' |
translate }}</b></button>
</div>
```

9.7.6. Liga

Componente que se muestra el menú principal de la liga

```
export class LeagueComponent implements OnInit {
 // Html
  public show: boolean;
  public option: string;
  public teams: boolean;
  public matchesUploaded: boolean;
  public finished: boolean;
  // Forms
  public journeysFormGroup: FormGroup;
  public informationFormGroup: FormGroup;
  public resultsFormGroup: FormGroup;
  // Get methods
  public competitionId: number;
  public competition$: Observable<Competition>;
  public competition: Competition;
  public information: string;
  public journeys = new Array<Journey>();
  public matchesJourneys: Match[][];
  //
  public countJourneys: number;
  public countCompleted: number;
  public notCompletedJourneys = new Array<Journey>();
  public journeyMatch = new Journey();

  public results: Array<any>;
  public winner: any;
  public newInformation: any;

  public matches = new Array<Match>();
  public matchGhost = new Match();

  public showMatches: any[][];
  public arrayMatch: Array<any>;
```

```
  public journeyIndex: number;
  public clicked: boolean;
  public break: number;
  public url: string;

  constructor(public alertService: AlertService,
    public utilsService: UtilsService,
    public loadingService: LoadingService,
    public groupService: GroupService,
    public translateService: TranslateService,
    public journeyService: JourneyService,
    public competitionService: CompetitionService,
    private route: ActivatedRoute,
    private router: Router,
    private matchesService: MatchesService,
    private _formBuilder: FormBuilder,
    private dialog?: MatDialog) {

      this.utilsService.currentUser =
Login.toObject(localStorage.getItem(AppConfig.LS_USER));
      this.utilsService.role = Number(localStorage.getItem(AppConfig.LS ROLE));
      this.option = 'Manualmente';
      this.matchesUploaded = false;
      this.finished = false;
      this.clicked = false;
      this.show = false;
      this.matchesJourneys = [];
      this.showMatches = [];
    }

  ngOnInit() {
    if ( this.utilsService.role === Role.TEACHER || this.utilsService.role ===
Role.STUDENT ) {
    this.loadingService.show();
    this.journeysFormGroup = this._formBuilder.group({
      id: ['', Validators.required],
      date: ['', Validators.required]
    });

    this.informationFormGroup = this. formBuilder.group({
      information: ['']
    });

    this.resultsFormGroup = this. formBuilder.group({
      results: this. formBuilder.array([
        this. formBuilder.group({
          winner: ['', Validators.required]
        })
      ])
    });

    this.competitionId = +this.route.snapshot.paramMap.get('id');
    this.getSelectedCompetition();
  }
  }

    getSelectedCompetition(): void {
      this.competition$ = this.competitionService.getCompetition(this.competitionId);
      this.competition$.subscribe(
        ((competition: Competition) => {
          this.competition = competition;
          this.information = competition.information;
          this.competition.mode === 'Equipos' ? this.teams = true : this.teams =
false;
          if (this.utilsService.role === Role.TEACHER) {
            this.getJourneys();
          } else {
            this.finished = true;
            this.loadingService.hide();
          }
        }),
        ((error: Response) => {
          this.loadingService.hide();
          this.alertService.show(error.toString());
        }));
    }
```

```
    getJourneys(): void {
      this.journeyService.getJourneysCompetition(this.competitionId).subscribe(
        ((journeys: Array<Journey>) => {
          this.journeys = journeys;
          this.journeys.sort(function (a, b) { return (a.number - b.number); });
          this.getMatches();
        }),
        ((error: Response) => {
          this.loadingService.hide();
          this.alertService.show(error.toString());
        }));
    }

    getMatches(): void {
      this.countJourneys = 0;
      for (let _n = 0; _n < this.journeys.length; _n++) {
        this.journeys[ n].completed = false;
        // Getting matches of each journey
        this.matchesJourneys[_n] = [];
        this.journeyService.getMatchesJourneyDetails(this.journeys[_n].id,
this.competition).subscribe(
        ((matches: Array<Match>) => {
          this.countCompleted = 0;
          this.countJourneys = this.countJourneys + 1;
          // Multidimensional array Journey[_n] and Matches[_m]
          for (let  m = 0;  m < matches.length;  m++) {
            this.matchesJourneys[ n][ m] = new Match();
            this.matchesJourneys[ n][ m] = matches[ m];
            if ( matches[_m].winner !== 0 ) { this.countCompleted++; }
          }
          // There are results for all matches so the journey is completed
          if ( this.countCompleted === matches.length ) {
            this.journeys[ n].completed = true;
          }
          // Making the array for journeys not completed
          if ( this.journeys[_n].completed === false) {
            this.notCompletedJourneys.push(this.journeys[_n]);
          }
          if ( this.countJourneys === this.journeys.length ) {
            this.notCompletedJourneys.sort(function (a, b) { return (a.number -
b.number); });
            this.finished = true;
            this.loadingService.hide();
          }
        }),
        ((error: Response) => {
          this.loadingService.hide();
          this.alertService.show(error.toString());
        }));
      }
    }


   loadResultSection() {
      if ( this.clicked === false ) {
        this.loadingService.show();
        this.clicked = true;
    // Searching the lower number of journey not completed
    if (this.notCompletedJourneys.length !== 0) {
      this.journeyMatch = this.notCompletedJourneys[0];
      for (let  j = 1;  j < this.notCompletedJourneys.length;  j++) {
        if ( this.notCompletedJourneys[ j].number < this.journeyMatch.number ) {
          this.journeyMatch = this.notCompletedJourneys[_j];
        }
      }
     this.journeyIndex = this.journeys.findIndex(((journey) => journey.id ===
this.journeyMatch.id));
     this.matches = this.matchesJourneys[this.journeyIndex];
      // Building matches to show
      for (let _m = 0; _m < this.matches.length; _m++) {
        if (this.matches[ m].namePlayerOne !== 'Ghost' &&
this.matches[ m].namePlayerTwo !== 'Ghost') {
        this.arrayMatch = [this.matches[ m].namePlayerOne,
        this.translateService.instant('CLASSIFICATION.DRAW2') ,
this.matches[_m].namePlayerTwo];
        this.showMatches[_m] = [];
```

```
        this.showMatches[ m] = this.arrayMatch;
       } else {
        this.break = _m;
        this.matchGhost = this.matches[_m];
      }
     }
     if (this.break !== undefined) {
     this.showMatches.splice(this.break, 1);
     }
       // Add the results of each match to section: introduce the results section
        for (let _a = 0; _a < this.showMatches.length - 1 ; _a++) {
        let results = <FormArray>this.resultsFormGroup.get('results');
        results.push(this. formBuilder.group({
          winner: ['', Validators.required]
        }));
        }

    }
    this.loadingService.hide();
    }
   }
    showInformation() {
      this.show === true ? this.show = false : this.show = true;
    }
    showResults() {
      this.option === 'Manualmente' ? this.option = 'Aleatoriamente' : this.option =
'Manualmente';
    }

    gotoClassification() {
      this.url = this.route.snapshot.url.join('/') + '/classification';
      this.router.navigate([this.url]);
    }

    gotoJourneys() {
      this.url = this.route.snapshot.url.join('/') + '/journeys';
      this.router.navigate([this.url]);
    }

    gotoTeams() {
      this.url = this.route.snapshot.url.join('/') + '/teams';
      this.router.navigate([this.url]);
    }

    onSubmitJourney (value) {
      this.loadingService.show();
      this.journeyService.putJourney(value).subscribe();
      this.loadingService.hide();

this.alertService.show(this.translateService.instant('COMPETITION CREATION.UPDATED JOU
RNEY'));
    }

    onSubmitInformation(value: string) {
     this.loadingService.show();
     this.competitionService.putInformation(value, this.competitionId).subscribe();
     this.newInformation = value;
     this.competition.information = this.newInformation.information;
     this.loadingService.hide();

this.alertService.show(this.translateService.instant('COMPETITION_CREATION.UPDATED_INF
ORMATION'));
    }

    onSubmitResults(value) {
      this.loadingService.show();
      if (value === undefined ) {
        this.results = [];
        for (let  m = 0;  m < this.showMatches.length;  m++) {
          this.results[_m] = {
            winner: this.showMatches[_m][Math.floor(Math.random() * 3) + 0]
          };
        }
        if ( this.matchGhost.playerOne === 0 || this.matchGhost.playerTwo === 0) {
          this.results.splice(this.break, 0, { winner: 'Descanso'} );
        }
      } else {
```

```
        this.results = value.results;
        if ( this.matchGhost.playerOne === 0 || this.matchGhost.playerTwo === 0) {
         this.results.splice(this.break, 0, {winner: 'Descanso'} );
        }
      }

      let numberPostMatches = 0;
      for (let _m = 0; _m < this.results.length; _m++) {
        this.winner = {winner: 0 };
        if ( this.matches[_m].namePlayerOne === this.results[_m].winner ) {
          this.winner.winner = this.matches[_m].playerOne;
        } else if (this.matches[ m].namePlayerTwo === this.results[ m].winner ) {
          this.winner.winner = this.matches[ m].playerTwo;
        } else if (this.translateService.instant('CLASSIFICATION.DRAW2') ===
this.results[_m].winner) {
          this.winner.winner = 1;
        } else if ('Descanso' === this.results[_m].winner) {
          this.winner.winner = 2;
        }
        this.matchesService.putWinner(this.winner, this.matches[_m].id)
        .subscribe( (match => {
          numberPostMatches ++;
          if (numberPostMatches === this.matchesJourneys[0].length ) {
            this.loadingService.hide();

this.alertService.show(this.translateService.instant('COMPETITION_CREATION.UPDATED_RES
ULTS'));
            this.matchesUploaded = true;
          }
        }),
        ((error: Response) => {
          this.loadingService.hide();
          this.alertService.show(error.toString());
        }));
      }
    }

    deleteCompetition() {
      const dialogRef = this.dialog.open(DeleteCompetitionComponent, {
        data: { competition: this.competition, journeys: this.journeys }
      });
    }

  }
```

```
<div *ngIf="competition" class="league-content">
  <div *ngIf="utilsService.role === 1 || utilsService.role === 0">
    <h5> {{ 'COMPETITIONS.TITLE' | translate }} > {{ competition.name }} </h5>

    <div *ngIf="finished">
    <div class="a-content">
     <a mat-raised-button class="classification" (click)="gotoClassification()">
       <strong>{{ 'CLASSIFICATION.TITLE' | translate }}</strong>
     </a>
    </div>

    <div class="a-content">
        <a mat-raised-button class="journeys" (click)="gotoJourneys()">
          <strong>{{ 'COMPETITIONS.JOURNEYS' | translate }}</strong>
        </a>
   </div>

    <div class="a-content" *ngIf="teams">
      <a mat-raised-button class="teams" (click)="gotoTeams()">
       <strong>{{ 'COMPETITIONS.TEAMS' | translate }}</strong>
      </a>
    </div>

   <div *ngIf="utilsService.role === 1" class="accordeon-journeys-info-results">

    <mat-accordion>

      <mat-expansion-panel class="option1">
        <mat-expansion-panel-header>
          <mat-panel-title>
            <strong  class="edit">{{ 'COMPETITIONS.MOD DATE' | translate }}</strong>
```

```
          </mat-panel-title>
          <mat-panel-description>
            {{ 'COMPETITIONS.MOD_DATE2' | translate }}
          </mat-panel-description>
        </mat-expansion-panel-header>

                <form [formGroup]="journeysFormGroup">

                    <div *ngIf="notCompletedJourneys.length === 0">
                        {{ 'COMPETITIONS.INTR_RESULT4' | translate }} {{
'COMPETITIONS.NO_MODIFY' | translate }}.
                    </div>
                    <div *ngIf="notCompletedJourneys.length !== 0">
                  <mat-form-field>
                      <mat-select #journeyId placeholder="{{
'COMPETITION_CREATION.JOURNEY' | translate }}"
                          formControlName="id" required>
                        <mat-option *ngFor="let journey of notCompletedJourneys"
[value]="journey.id">
                          {{ journey.number }}
                        </mat-option>
                      </mat-select>
                  </mat-form-field>

                  <mat-form-field>
                    <input matInput [matDatepicker]="picker" formControlName="date"
                      placeholder="{{ 'COMPETITION CREATION.DATE' | translate }}"
required>
                      <mat-datepicker-toggle matSuffix [for]="picker"></mat-
datepicker-toggle>
                      <mat-datepicker #picker></mat-datepicker>
                  </mat-form-field>
                  </div>
                  <div *ngIf="notCompletedJourneys.length !== 0">
                    <button mat-raised-button class="edit-btn"
(click)="onSubmitJourney(journeysFormGroup.value)"
[disabled]="!journeysFormGroup.valid">
                        <b>{{ 'COMPETITIONS.MODIFY' | translate }}</b>
                    </button>
                  </div>
                </form>
            </mat-expansion-panel>

            <mat-expansion-panel class="option2">
                <mat-expansion-panel-header>
                  <mat-panel-title>
                      <strong class="edit">{{ 'COMPETITIONS.MOD TEXT' | translate
}}</strong>
                  </mat-panel-title>
                  <mat-panel-description>
                    {{ 'COMPETITIONS.MOD TEXT2' | translate }}
                  </mat-panel-description>
                </mat-expansion-panel-header>

                <form [formGroup]="informationFormGroup">
                <mat-form-field class="text-info">
                    <textarea matInput [value]="information"
                     formControlName="information" required></textarea>
                </mat-form-field>
                <div>
                    <button mat-raised-button class="edit-btn"
(click)="onSubmitInformation(informationFormGroup.value)"
[disabled]="!informationFormGroup.valid">
                        <b>{{ 'COMPETITIONS.MODIFY' | translate }}</b>
                    </button>
                </div>
                 </form>
              </mat-expansion-panel>

      <mat-expansion-panel class="option3">
        <mat-expansion-panel-header (click)="loadResultSection()">
          <mat-panel-title>
           <b>{{ 'COMPETITIONS.INTR RESULT' | translate }}</b>
          </mat-panel-title>
          <mat-panel-description>
           {{ 'COMPETITIONS.INTR_RESULT2' | translate }}
          </mat-panel-description>
```

```
        </mat-expansion-panel-header>
        <div *ngIf="notCompletedJourneys.length !== 0">
        <div *ngIf=!matchesUploaded>
          <p><b>¿{{ 'COMPETITIONS.QUESTION_RESULTS' | translate }} {{
journeyMatch.number }}?</b></p>
          <button [disabled]="option === 'Manualmente'" class="btn-rnd" mat-raised-
button (click)="showResults()">
            <b>{{ 'COMPETITIONS.MANUALLY' | translate }}</b>
          </button>
          <button [disabled]="option === 'Aleatoriamente'" class="btn-rnd" mat-raised-
button (click)="showResults()">
            <b>{{ 'COMPETITIONS.RANDOM' | translate }}</b>
          </button>
          <p>{{ 'COMPETITIONS.SELECTED' | translate }}: <b>{{option}}</b> </p>

        <mat-card *ngIf="option === 'Manualmente'">
           <div *ngIf="competition.mode === 'Individual'"><p><b>{{
'COMPETITIONS.INTR RESULT3' | translate }} {{ journeyMatch.number }}:</b></p></div>
           <div *ngIf="competition.mode === 'Equipos'"><p><b>{{
'COMPETITIONS.INTR_RESULT3_TEAM' | translate }} {{ journeyMatch.number
}}:</b></p></div>
           <form [formGroup]="resultsFormGroup">
             <div class="form-group" formArrayName="results">
               <div *ngFor="let match of resultsFormGroup.controls.results.controls
;let i=index;">
                <div class="panel-body" [formGroupName]="i">
                   <mat-radio-group  formControlName="winner" required>
                     <div>
                       <mat-radio-button class="match{{n}}" [value]="match"
*ngFor="let match of showMatches[i];let n=index;">
                          {{ match }}
                       </mat-radio-button>
                     </div>
                   </mat-radio-group>
                </div>
               </div>
             </div>

             <div>
               <button mat-raised-button class="edit-btn"
(click)="onSubmitResults(resultsFormGroup.value)"
[disabled]="!resultsFormGroup.valid">
                  <b>{{ 'COMPETITIONS.INTR_RESULT1' | translate }}</b>
               </button>
             </div>
           </form>
         </mat-card >
         <mat-card  *ngIf="option === 'Aleatoriamente'">
            <p><b>{{ 'COMPETITIONS.RANDOM2' | translate }} {{ journeyMatch.number }}
{{ 'COMPETITIONS.RANDOM3' | translate }} {{ 'COMPETITIONS.INTR RESULT1' | translate
}}:</b></p>
            <button mat-raised-button class="edit-btn" (click)="onSubmitResults()">
               <b>{{ 'COMPETITIONS.INTR_RESULT1' | translate }}</b>
            </button>
         </mat-card >
      </div>
      <div *ngIf=matchesUploaded>
          {{ 'COMPETITIONS.INTR RESULT5' | translate }} {{ journeyMatch.number }} {{
'COMPETITIONS.RESULTS_SUCCESS' | translate }}
      </div>
      </div>

      <div *ngIf="notCompletedJourneys.length === 0">
       {{ 'COMPETITIONS.INTR_RESULT4' | translate }}
      </div>
      </mat-expansion-panel>

    </mat-accordion>

    </div>

    <div class="show-info-content">
        <button *ngIf="!show"  mat-raised-button (click)="showInformation()">
            <mat-icon>info</mat-icon>
            <b>{{ 'COMPETITIONS.SHOW_INFO' | translate }}</b>
        </button>
        <button *ngIf="show"  mat-raised-button (click)="showInformation()">
```

```
            <mat-icon>info outline</mat-icon>
            <b>{{ 'COMPETITIONS.HIDE_INFO' | translate }}</b>
        </button>
        <mat-card *ngIf="show" class="information">
            <mat-card-header>
              <mat-card-title><b>{{ 'COMPETITIONS.INFO COMP' | translate }}: <span
class="competi-name">{{competition.name}}</span></b></mat-card-title>
              <mat-card-subtitle>{{ 'COMPETITIONS.TYPE' | translate }}:
{{competition.type}}, {{ 'COMPETITIONS.MODE' | translate }}:
{{competition.mode}}</mat-card-subtitle>
            </mat-card-header>
            <mat-card-content>
                {{competition.information}}
            </mat-card-content>
          </mat-card>
    </div>

    <div class="a-content" *ngIf="utilsService.role === 0">
      <a mat-raised-button>
        <strong>{{ 'COMPETITIONS.PLAY' | translate }}</strong>
      </a>
    </div>
    <div class="a-content" *ngIf="utilsService.role === 1">
      <button mat-raised-button class="delete" (click)="deleteCompetition()">
        <mat-icon>delete</mat-icon>
        <strong>{{ 'COMPETITIONS.DELETE' | translate }}</strong>
      </button>
    </div>

    </div>

    <div class="final-button">
        <a mat-button [routerLink]="'/competitions'">
          <mat-icon mat-list-icon>navigate before</mat-icon>
          {{ 'COMPETITIONS.RETURN2' | translate }}
        </a>
    </div>

  </div>
  </div>
```

9.7.7. Clasificación de la liga

```
export class ClassificationComponent implements OnInit {

  public competition: Competition;
  public competitionId: string;
  public modeIndividual: boolean;

  public journeys: Journey[];
  public participants: Participant[];
  public odd: boolean;
  public matchesJourneys: Array<Array<Match>>;

  public scores = new Array<Score>();
  public score: Score;

  constructor(public utilsService: UtilsService,
    public loadingService: LoadingService,
    public alertService: AlertService,
    public journeyService: JourneyService,
    private route: ActivatedRoute,
    private teamService: TeamService,
    private competitionService: CompetitionService) {
      this.utilsService.currentUser =
Login.toObject(localStorage.getItem(AppConfig.LS_USER));
      this.utilsService.role = Number(localStorage.getItem(AppConfig.LS_ROLE));
     }

  ngOnInit() {
    if (this.utilsService.role === Role.TEACHER || this.utilsService.role ===
Role.STUDENT) {
    this.loadingService.show();
```

```
    this.competitionId = this.route.snapshot.paramMap.get('id');
    this.getClassificationOfCompetition();
    }
  }

  getClassificationOfCompetition(): void {
    this.competitionService.getCompetition(this.competitionId)
    .subscribe( ((competition: Competition) => {
        this.competition = competition;
        this.getJourneys();
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
  }

  getJourneys(): void {
    this.journeyService.getJourneysCompetition(this.competitionId).subscribe(
      ((journeys: Array<Journey>) => {
        this.journeys = journeys;
        this.getMatches();
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
  }

  getMatches(): void {
    let countJourneys = 0;
    this.matchesJourneys = [];
    for (let  j = 0;  j < this.journeys.length;  j++) {
      this.matchesJourneys[ j] = [];
      this.journeyService.getMatchesJourneyDetails(this.journeys[ j].id,
this.competition).subscribe(
      ((matches: Array<Match>) => {
        countJourneys = countJourneys + 1;
        for (let  m = 0;  m < matches.length;  m++) {
          this.matchesJourneys[ j][ m] = new Match();
          this.matchesJourneys[ j][ m] = matches[ m];
        }
         if ( countJourneys === this.journeys.length ) {
           this.getParticipants();
          }
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
    }
  }

  getParticipants(): void {
    this.participants = [];
    if (this.competition.mode === 'Individual') {
      this.modeIndividual = true;
      this.competitionService.getStudentsCompetition(this.competitionId)
      .subscribe(( (students: Array<Student>) => {
        if (students.length % 2 === 0 ) { this.odd = false; } else { this.odd = true;
}
        for (let  s = 0;  s < students.length;  s++) {
          this.participants[ s] = {
            id: +students[_s].id,
            name: students[_s].name.concat(' ', students[_s].surname)
          };
        }
        this.getScores();
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
      } else {
      this.modeIndividual = false;
      this.teamService.getTeamsCompetition(this.competitionId)
      .subscribe(( (teams: Array<Team>) => {
```

```
        if (teams.length % 2 === 0 ) { this.odd = false; } else { this.odd = true; }
        for (let _t = 0; _t < teams.length; _t++) {
          this.participants[_t] = {
            id: +teams[_t].id,
            name: teams[_t].name
          };
        }
        this.getScores();
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
      }
  }

  getScores(): void {
      this.scores = [];
      for (let  p = 0;  p < this.participants.length;  p++) {
        this.score = { position: 0, name: this.participants[_p].name,
                       played: 0, won: 0, draw: 0, lost: 0, points: 0};
        for (let  j = 0;  j < this.journeys.length;  j++) {
          let found = false;
          for (let  m = 0;  m < this.matchesJourneys[ j].length && !found;  m++) {
            if ( +this.participants[_p].id === this.matchesJourneys[_j][_m].playerOne
||
            +this.participants[ p].id === this.matchesJourneys[ j][ m].playerTwo ) {
              if ( this.matchesJourneys[ j][ m].winner === +this.participants[ p].id )
{
                this.score.points = this.score.points + 3;
                this.score.won = this.score.won + 1;
                this.score.played = this.score.played + 1;
              } else if ( this.matchesJourneys[ j][ m].winner === 1 ) {
                this.score.points = this.score.points + 1;
                this.score.draw = this.score.draw + 1;
                this.score.played = this.score.played + 1;
              } else if ( this.matchesJourneys[_j][_m].winner === 2
              || this.matchesJourneys[_j][_m].winner === 0 ) {
              } else {
                this.score.lost = this.score.lost + 1;
                this.score.played = this.score.played + 1;
              }
              found = true;
            }
          }
        }
        this.scores.push(this.score);
      }
      this.scores.sort(function (a, b) {
        return (b.points - a.points);
      });
      for (let  s = 0;  s < this.scores.length;  s++) {
        this.scores[_s].position = _s + 1;
       }
       this.loadingService.hide();
  }
}

export interface Score {
  position: number;
  name: string;
  played: number;
  won: number;
  draw: number;
  lost: number;
  points: number;
}

export interface Participant {
  id: number;
  name: string;
}
```

```
<div *ngIf="utilsService.role === 1 || utilsService.role === 0" class="classification-
content">
```

```
    <h5   *ngIf="competition">   {{   'COMPETITIONS.TITLE'   |   translate   }}   >   {{
competition.name }} >
      {{ 'CLASSIFICATION.TITLE' | translate }} </h5>

  <mat-card>
    <table>
      <caption   *ngIf="competition">{{   'CLASSIFICATION.LEAGUE'   |   translate   }}
{{competition.name}}</caption>
      <thead>
        <tr class="tableHeader">
          <th class="position">{{ 'CLASSIFICATION.N' | translate }}</th>
          <th   *ngIf="modeIndividual"   class="name">{{   'CLASSIFICATION.STUDENT'   |
translate }}</th>
          <th *ngIf="!modeIndividual"class="name">{{ 'CLASSIFICATION.TEAM' | translate
}}</th>
          <th>{{ 'CLASSIFICATION.PLAYED' | translate }}</th>
          <th>{{ 'CLASSIFICATION.WON' | translate }}</th>
          <th>{{ 'CLASSIFICATION.DRAW' | translate }}</th>
          <th>{{ 'CLASSIFICATION.LOST' | translate }}</th>
          <th>{{ 'CLASSIFICATION.POINTS' | translate }}</th>
        </tr>
      </thead>
      <tbody>
        <tr *ngFor="let score of scores ; let i=index;">
          <td id="position_{{i + 1}}" class="position">{{score.position}}</td>
          <td class="name">{{score.name}}</td>
          <td>{{score.played}}</td>
          <td>{{score.won}}</td>
          <td>{{score.draw}}</td>
          <td>{{score.lost}}</td>
          <td class="points">{{score.points}}</td>
        </tr>
      </tbody>
    </table>
    <div *ngIf="odd" class="information">
        {{ 'CLASSIFICATION.INFORMATION' | translate }}
    </div>
  </mat-card>
    <div class="final-buttons">
        <a mat-button [routerLink]="['/competition/league', competitionId]">
          <mat-icon mat-list-icon>navigate before</mat-icon>
          {{ 'TEAMS.RETURN' | translate }}
        </a>
    </div>
</div>
```

9.7.8. Calendario de la liga

```
export class JourneysLeagueComponent implements OnInit {

  public competitionId: string;
  public competition: Competition;
  public journeys = new Array<Journey>();
  public dates: String[];
  public matchesJourneys: Match[][];
  public matches: Match[];
  public descanso: number;

  constructor(public alertService: AlertService,
    public utilsService: UtilsService,
    public loadingService: LoadingService,
    public translateService: TranslateService,
    public competitionService: CompetitionService,
    public journeyService: JourneyService,
    public teamService: TeamService,
    public datePipe: DatePipe,
    private route: ActivatedRoute) {
      this.utilsService.currentUser =
Login.toObject(localStorage.getItem(AppConfig.LS USER));
      this.utilsService.role = Number(localStorage.getItem(AppConfig.LS ROLE));
    }

  ngOnInit() {
```

```
    if (this.utilsService.role === Role.TEACHER || this.utilsService.role ===
Role.STUDENT) {
    this.loadingService.show();
    this.competitionId = this.route.snapshot.paramMap.get('id');
    this.getSelectedCompetition();
    }
  }

  getSelectedCompetition(): void {
    this.competitionService.getCompetition(this.competitionId).subscribe(
      ((competition: Competition) => {
        this.competition = competition;
        this.getJourneys();
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
  }

  getJourneys(): void {
    this.journeyService.getJourneysCompetition(this.competitionId).subscribe(
      ((journeys: Array<Journey>) => {
        this.journeys = journeys;
        this.journeys.sort(function (a, b) {
          return (a.number - b.number);
        });
        this.getMatches();
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
  }

  getMatches(): void {
    let countJourneys = 0;
    this.matchesJourneys = [];
    for (let  j = 0;  j < this.journeys.length;  j++) {
      this.matchesJourneys[ j] = [];
      this.journeyService.getMatchesJourneyDetails(this.journeys[ j].id,
this.competition).subscribe(
      ((matches: Array<Match>) => {
        countJourneys = countJourneys + 1;
        this.matches = matches;
        for (let  m = 0;  m < this.matches.length;  m++) {
          if (this.matches[ m].namePlayerOne === 'Ghost' ||
this.matches[_m].namePlayerTwo === 'Ghost') {
            this.descanso = _m;
          }
        }
        if (this.descanso !== undefined) {
          this.matches.splice(this.descanso, 1);
          }
        this.matchesJourneys[_j] = this.matches;
         if ( countJourneys === this.journeys.length ) {
           this.getDatesAndResults();
          }
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
    }
  }

  getDatesAndResults(): void {
    this.dates = [];
    for (let  j = 0;  j < this.journeys.length;  j++) {
      if (this.journeys[_j].date === null) {
        this.dates[_j] =
this.translateService.instant('COMPETITIONS.NOT ESTABLISHED');
      } else {
        this.dates[ j] = this.datePipe.transform(this.journeys[ j].date, 'dd-MM-
yyyy');
      }
      for (let _m = 0; _m < this.matchesJourneys[_j].length; _m++) {
```

```
        if (this.matchesJourneys[ j][ m].winner ===
this.matchesJourneys[_j][_m].playerOne ) {
          this.matchesJourneys[_j][_m].result =
this.matchesJourneys[_j][_m].namePlayerOne;
        } else if ( this.matchesJourneys[_j][_m].winner ===
this.matchesJourneys[ j][ m].playerTwo ) {
          this.matchesJourneys[ j][ m].result =
this.matchesJourneys[_j][_m].namePlayerTwo;
        } else if ( this.matchesJourneys[_j][_m].winner === 1 ) {
          this.matchesJourneys[_j][_m].result =
this.translateService.instant('CLASSIFICATION.DRAW2');
        } else if ( this.matchesJourneys[ j][ m].winner === 0 ) {
          this.matchesJourneys[ j][ m].result = '-';
        }
      }
    }
    this.loadingService.hide();
  }
}
```

```
<div *ngIf="utilsService.role === 1 || utilsService.role === 0" class="journeys-
content">
  <h5 *ngIf="competition"> {{ 'COMPETITIONS.TITLE' | translate }} > {{
competition.name }} >
        {{ 'COMPETITIONS.CALENDAR' | translate }} </h5>

  <div *ngIf="dates">
  <h4>{{ 'COMPETITIONS.CALENDAR OF' | translate }} {{ 'COMPETITIONS.LEAGUE' |
translate }}: {{ competition.name }}</h4>
      <mat-card class="journey-card" *ngFor="let date of dates; let i=index;">
        <mat-card-content>
            <p>{{ 'COMPETITIONS.JOURNEY' | translate }} {{i + 1}}</p>
         <table>
            <caption>
              {{ 'COMPETITIONS.DATE' | translate }}: {{date}}
            </caption>
            <thead>
              <tr class="tableHeader">
                <th *ngIf="competition.mode === 'Individual'" class="player">{{
'COMPETITIONS.PLAYER' | translate }} 1</th>
                <th *ngIf="competition.mode === 'Individual'" class="player">{{
'COMPETITIONS.PLAYER' | translate }} 2</th>
                <th *ngIf="competition.mode === 'Equipos'" class="player">{{
'COMPETITIONS.TEAM' | translate }} 1</th>
                <th *ngIf="competition.mode === 'Equipos'" class="player">{{
'COMPETITIONS.TEAM' | translate }} 2</th>
                <th class="result">{{ 'COMPETITIONS.WINNER' | translate }}</th>
              </tr>
            </thead>
            <tbody>
              <tr *ngFor="let score of matchesJourneys[i]">
                <td class="player">{{score.namePlayerOne}}</td>
                <td class="player">{{score.namePlayerTwo}}</td>
                <td class="result">{{score.result}}</td>
              </tr>
            </tbody>
          </table>

        </mat-card-content>
      </mat-card>
  </div>

  <div class="final-buttons">
      <a mat-button [routerLink]="['/competition/league', competitionId]">
        <mat-icon mat-list-icon>navigate before</mat-icon>
        {{ 'COMPETITIONS.RETURN' | translate }}
      </a>
  </div>

</div>
```

9.7.9. Torneo de tenis

Componente que se muestra el menú principal del torneo de tenis

```
export class TennisComponent implements OnInit {

  // Html
  public show: boolean; // information
  public option: string; // random or manually
  public matchesUploaded: boolean;
  public finished: boolean;
  // Forms
  public journeysFormGroup: FormGroup;
  public informationFormGroup: FormGroup;
  public resultsFormGroup: FormGroup;
  // Get methods
  public competitionId: number;
  public competition: Competition;
  public information: string;
  public journeys = new Array<Journey>();
  public matchesJourneys: Match[][];
  public lastJourney: number;
  // Expansion panels
  public notCompletedJourneys = new Array<Journey>();
  public newInformation: any;
  public clicked: boolean;
  public matches = new Array<Match>();
  public showMatchesPrimary: String[][];
  public showMatchesIdPrimary: Number[][];
  public ghostsPrimary: Array<Array<number>>;
  public showMatchesSecondary: String[][];
  public showMatchesIdSecondary: Number[][];
  public ghostsSecondary: Array<Array<number>>;
  // Submit results
  public results: any[];
  public results2: any[];
  public matchesIdPrimary: Array<number>;
  public matchesIdSecondary: Array<number>;
  // Next journey
  public postMatches: Array<Match>;
  public principalMatches: Array<number>;
  public secondaryMatches: Array<number>;
  public submitMatches: Array<number>;
  public match1: any;

  constructor(public alertService: AlertService,
    public utilsService: UtilsService,
    public loadingService: LoadingService,
    public translateService: TranslateService,
    public groupService: GroupService,
    public journeyService: JourneyService,
    public competitionService: CompetitionService,
    private route: ActivatedRoute,
    private router: Router,
    private matchesService: MatchesService,
    private  formBuilder: FormBuilder,
    private dialog?: MatDialog) {
      this.utilsService.currentUser =
Login.toObject(localStorage.getItem(AppConfig.LS_USER));
      this.utilsService.role = Number(localStorage.getItem(AppConfig.LS_ROLE));
      this.show = false;
      this.option = 'Manualmente';
      this.matchesUploaded = false;
      this.finished = false;
      this.clicked = false;
     }

  ngOnInit() {
    if ( this.utilsService.role === Role.TEACHER || this.utilsService.role ===
Role.STUDENT ) {
    this.loadingService.show();
    this.journeysFormGroup = this._formBuilder.group({
      id: ['', Validators.required],
      date: ['', Validators.required]
    });

    this.informationFormGroup = this._formBuilder.group({
      information: ['']
```

```
    });

    this.resultsFormGroup = this._formBuilder.group({
      results: this._formBuilder.array([
        this._formBuilder.group({
          winner: ['', Validators.required]
        })
      ]),
      results2: this._formBuilder.array([
        this._formBuilder.group({
          winner: ['', Validators.required]
        })
      ])
    });

    this.competitionId = +this.route.snapshot.paramMap.get('id');
    this.getSelectedCompetition();
  }
  }

  getSelectedCompetition(): void {
    this.competitionService.getCompetition(this.competitionId).subscribe(
      ((competition: Competition) => {
        this.competition = competition;
        this.information = competition.information;
        if (this.utilsService.role === Role.TEACHER) {
          this.getJourneys();
        } else {
          this.finished = true;
          this.loadingService.hide();
        }
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
  }

  getJourneys(): void {
    this.journeyService.getJourneysCompetition(this.competitionId).subscribe(
      ((journeys: Array<Journey>) => {
        this.journeys = journeys;
        this.journeys.sort(function (a, b) { return (a.number - b.number); });
        this.getMatches();
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
  }

  getMatches(): void {
    this.matchesJourneys = [];
    let journeysCount = 0;
    for (let _n = 0; _n < this.journeys.length; _n++) {
      this.journeys[ n].completed = false;
      this.journeyService.getMatchesJourneyDetails(this.journeys[ n].id,
this.competition).subscribe(
      ((matches: Array<Match>) => {
        this.matchesJourneys[_n] = [];
        let incompletedJourney = 0;
        for (let  m = 0;  m < matches.length;  m++) {
          this.matchesJourneys[ n][ m] = new Match();
          this.matchesJourneys[_n][_m] = matches[_m];
          if (this.matchesJourneys[_n][_m].winner === 0) {
            incompletedJourney++;
            if ( incompletedJourney === matches.length ) {
              this.lastJourney =  n;
              this.matches = matches;
            }
          }
        }
        journeysCount++;
        if ( journeysCount === this.lastJourney + 1 || journeysCount ===
this.journeys.length) {
          this.loadJourneysSection();
        }
```

```
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
    }
  }

  loadJourneysSection() {
    for (let _j = this.lastJourney; _j < this.journeys.length; _j++) {
this.notCompletedJourneys.push(this.journeys[_j]); }
    this.finished = true;
    this.loadingService.hide();
  }

  loadResultSection() {
    if (!this.clicked) {
      this.clicked = true;
      this.loadingService.show();
      // Building matches to show
      this.showMatchesPrimary = [];
      this.showMatchesIdPrimary = [];
      this.ghostsPrimary = [];
      this.matchesIdPrimary = [];
      this.showMatchesSecondary = [];
      this.showMatchesIdSecondary = [];
      this.ghostsSecondary = [];
      this.matchesIdSecondary = [];
      if ((this.lastJourney + 1) % 2 !== 0) { // jornada impar
        if (this.lastJourney === 0) {
          for (let _m = 0; _m < this.matches.length; _m++) {
            if (this.matches[_m].namePlayerOne !== 'Ghost' &&
this.matches[ m].namePlayerTwo !== 'Ghost') {
            this.showMatchesIdPrimary.push([this.matches[ m].playerOne,
this.matches[ m].playerTwo]);
            this.showMatchesPrimary.push([this.matches[_m].namePlayerOne,
this.matches[_m].namePlayerTwo]);
            this.matchesIdPrimary.push(+this.matches[_m].id);
            } else {
            this.ghostsPrimary.push([this.matches[ m].playerOne,
this.matches[ m].playerTwo, +this.matches[ m].id]);
            }
          }
        } else {
          for (let  m = 0;  m < this.matches.length;  m++) {
            if (this.matches[ m].namePlayerOne !== 'Ghost' &&
this.matches[ m].namePlayerTwo !== 'Ghost') {
            this.showMatchesIdSecondary.push([this.matches[_m].playerOne,
this.matches[_m].playerTwo]);
            this.showMatchesSecondary.push([this.matches[ m].namePlayerOne,
this.matches[ m].namePlayerTwo]);
            this.matchesIdSecondary.push(+this.matches[ m].id);
            } else {
            this.ghostsSecondary.push([this.matches[_m].playerOne,
this.matches[_m].playerTwo, +this.matches[_m].id]);
            }
          }
        }
    } else {   // journeys with principal and secondary matches
        for (let _m = 0; _m < this.matches.length; _m++) {
          if ( _m < this.matches.length / 2 ) {
            if (this.matches[ m].namePlayerOne !== 'Ghost' &&
this.matches[ m].namePlayerTwo !== 'Ghost') {
            this.showMatchesIdPrimary.push([this.matches[_m].playerOne,
this.matches[_m].playerTwo]);
            this.showMatchesPrimary.push([this.matches[_m].namePlayerOne,
this.matches[_m].namePlayerTwo]);
            this.matchesIdPrimary.push(+this.matches[ m].id);
            } else {
            this.ghostsPrimary.push([this.matches[_m].playerOne,
this.matches[_m].playerTwo, +this.matches[_m].id]);
            }
          } else {
            if (this.matches[ m].namePlayerOne !== 'Ghost' &&
this.matches[_m].namePlayerTwo !== 'Ghost') {
              this.showMatchesIdSecondary.push([this.matches[_m].playerOne,
this.matches[_m].playerTwo]);
```

```
              this.showMatchesSecondary.push([this.matches[ m].namePlayerOne,
this.matches[_m].namePlayerTwo]);
              this.matchesIdSecondary.push(+this.matches[_m].id);
            } else {
              this.ghostsSecondary.push([this.matches[_m].playerOne,
this.matches[ m].playerTwo, +this.matches[ m].id]);
            }
          }
        }
       }
       for (let _a = 0; _a < this.showMatchesPrimary.length - 1 ; _a++) {
        let results = <FormArray>this.resultsFormGroup.get('results');
        results.push(this. formBuilder.group({
          winner: ['', Validators.required]
        }));
      }
      for (let _a = 0; _a < this.showMatchesSecondary.length - 1 ; _a++) {
        let results2 = <FormArray>this.resultsFormGroup.get('results2');
        results2.push(this. formBuilder.group({
          winner: ['', Validators.required]
        }));
      }
      if (this.showMatchesPrimary.length === 0) {
        let primary = <FormArray>this.resultsFormGroup.get('results');
        primary.removeAt(0);
       }
      if (this.showMatchesSecondary.length === 0) {
        let secondary = <FormArray>this.resultsFormGroup.get('results2');
        secondary.removeAt(0);
      }
  }
  this.loadingService.hide();
  }

  onSubmitJourney (value) {
    this.loadingService.show();
    this.journeyService.putJourney(value).subscribe();
    this.loadingService.hide();

this.alertService.show(this.translateService.instant('COMPETITION CREATION.UPDATED JOU
RNEY'));
  }

  onSubmitInformation(value: string) {
    this.loadingService.show();
    this.competitionService.putInformation(value, this.competitionId).subscribe();
    this.newInformation = value;
    this.competition.information = this.newInformation.information;
    this.loadingService.hide();

this.alertService.show(this.translateService.instant('COMPETITION CREATION.UPDATED INF
ORMATION'));
  }

  onSubmitResults(value) {
    this.loadingService.show();
    this.results = [];
    this.results2 = [];
    this.postMatches = [];

    if (value === undefined ) {
      for (let  m = 0;  m < this.showMatchesIdPrimary.length;  m++) {
        this.results[ m] = {
          winner: this.showMatchesIdPrimary[_m][Math.floor(Math.random() * 2) + 0]
        };
      }
       // adding ghosts of principal tournament with the winner no ghost
      for (let  g = 0;  g < this.ghostsPrimary.length;  g++) {
       this.ghostsPrimary[ g][0] === 0 ?
       this.results.push({ winner: this.ghostsPrimary[_g][1] }) : this.results.push({
winner: this.ghostsPrimary[_g][0] });
       this.matchesIdPrimary.push(this.ghostsPrimary[ g][2]); // adding id ghost
matches
      }
      // The same with the secondary tournament
      for (let _v = 0; _v < this.showMatchesIdSecondary.length; _v++) {
        this.results2[_v] = {
```

```
          winner: this.showMatchesIdSecondary[ v][Math.floor(Math.random() * 2) + 0]
        };
      }
      for (let _g = 0; _g < this.ghostsSecondary.length; _g++) {
        this.ghostsSecondary[_g][0] === 0 ?
        this.results2.push({ winner: this.ghostsSecondary[ g][1] }) :
this.results2.push({ winner: this.ghostsSecondary[ g][0] });
        this.matchesIdSecondary.push(this.ghostsSecondary[_g][2]); // id ghost mtches
      }
    } else {
      // converts the name: { winner: 10000 }
      for (let  v = 0;  v < value.results.length;  v++) {
        let index = this.showMatchesPrimary[ v].indexOf(value.results[ v].winner);
        this.results[_v] = { winner: this.showMatchesIdPrimary[_v][index] };
      }
      // adding ghosts of principal tournament with the winner no ghost
      for (let _g = 0; _g < this.ghostsPrimary.length; _g++) {
      this.ghostsPrimary[ g][0] === 0 ?
      this.results.push({ winner: this.ghostsPrimary[ g][1] }) : this.results.push({
winner: this.ghostsPrimary[_g][0] });
      this.matchesIdPrimary.push(this.ghostsPrimary[_g][2]); // id ghost matches
      }
      // here we have the firt part with and without ghosts

      // the same for the secondary tournament
      for (let _v = 0; _v < value.results2.length; _v++) {
      let index = this.showMatchesSecondary[ v].indexOf(value.results2[ v].winner);
      this.results2[ v] = { winner: this.showMatchesIdSecondary[ v][index] };
      }
      for (let _g = 0; _g < this.ghostsSecondary.length; _g++) {
      this.ghostsSecondary[_g][0] === 0 ?
      this.results2.push({ winner: this.ghostsSecondary[_g][1] }) :
this.results2.push({ winner: this.ghostsSecondary[ g][0] });
      this.matchesIdSecondary.push(this.ghostsSecondary[ g][2]); // id ghost matches
      }
    }

   const allResults = this.results.concat(this.results2);
   const allMatches = this.matchesIdPrimary.concat(this.matchesIdSecondary);

    for (let  m = 0;  m < allResults.length;  m++) {
      this.matchesService.putWinner(allResults[_m], allMatches[_m])
      .subscribe( (match => {
        this.postMatches.push(match);
        if (this.postMatches.length === allResults.length ) {
        this.postMatches.sort(function (a, b) { return (+a.id - +b.id); });

this.alertService.show(this.translateService.instant('COMPETITION_CREATION.UPDATED_RES
ULTS'));
        this.matchesUploaded = true;
        if ((this.lastJourney + 1) !== this.journeys.length) {
          this.postNextJourney();
        } else { this.loadingService.hide(); }
        }
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
       }));
    }
  }

  postNextJourney() {
    this.principalMatches = [];
    this.secondaryMatches = [];
    this.submitMatches = [];
    if ((this.lastJourney + 1) % 2 !== 0) { // odd journey: the next journey will have
2 tournaments
     if (this.lastJourney === 0) {
      for (let _m = 0; _m < this.postMatches.length; _m++) {
        if (this.postMatches[_m].winner === this.postMatches[_m].playerOne) {
          this.principalMatches.push(this.postMatches[ m].playerOne);
          this.secondaryMatches.push(this.postMatches[ m].playerTwo);
          } else {
          this.principalMatches.push(this.postMatches[_m].playerTwo);
          this.secondaryMatches.push(this.postMatches[_m].playerOne);
        }
```

```
      }
     } else {
      for (let _m = 0; _m < (this.matchesJourneys[this.lastJourney - 1].length / 2);
_m++) {
        this.matchesJourneys[this.lastJourney - 1][_m].winner ===
this.matchesJourneys[this.lastJourney - 1][ m].playerOne ?
        this.principalMatches.push(this.matchesJourneys[this.lastJourney -
1][_m].playerOne) :
        this.principalMatches.push(this.matchesJourneys[this.lastJourney -
1][_m].playerTwo);
      }
      for (let  m = 0;  m < this.postMatches.length;  m++) {
        this.postMatches[ m].winner === this.postMatches[ m].playerOne ?
        this.secondaryMatches.push(this.postMatches[_m].playerOne) :
        this.secondaryMatches.push(this.postMatches[_m].playerTwo);
      }
     }
      this.submitMatches = this.principalMatches.concat(this.secondaryMatches);
    } else if ( (this.lastJourney + 1) % 2 === 0 ) {
      for (let _m = 0; _m < this.postMatches.length; _m++) {
        if ( _m < (this.postMatches.length / 2)) {
          this.postMatches[ m].winner === this.postMatches[ m].playerOne ?
            this.secondaryMatches.push(this.postMatches[ m].playerTwo) :
            this.secondaryMatches.push(this.postMatches[ m].playerOne);
        } else {
          this.postMatches[_m].winner === this.postMatches[_m].playerOne ?
            this.secondaryMatches.push(this.postMatches[ m].playerOne) :
            this.secondaryMatches.push(this.postMatches[ m].playerTwo);
        }
      }
      this.submitMatches = this.secondaryMatches;
    }

    let countMatches = 0;

    for (let _s = 0; _s < this.secondaryMatches.length; _s += 2) {
     this.match1 = {
       playerOne : +this.submitMatches[_s],
       playerTwo : +this.submitMatches[ s + 1],
       journeyId : +this.journeys[this.lastJourney + 1].id
     };
       // POST MATCHES
       this.matchesService.postMatch(this.match1)
       .subscribe( (match => {
         countMatches++;
         if ( countMatches === (this.secondaryMatches.length / 2)) {
          if ((this.lastJourney + 1) % 2 !== 0) {
            this.postSecondaryTournament();
          } else {
            this.loadingService.hide();

this.alertService.show(this.translateService.instant('TENNIS.UPDATED MATCHES'));
          }
         }
         if ((this.lastJourney + 1) === (this.journeys.length - 1)) {
          this.loadingService.hide();

this.alertService.show(this.translateService.instant('TENNIS.UPDATED MATCH'));
         }
       }),
       ((error: Response) => {
         this.loadingService.hide();
         this.alertService.show(error.toString());
       }));
    }
   }

   postSecondaryTournament() {
    let countMatches = 0;
    for (let _s = 0; _s < this.secondaryMatches.length; _s += 2) {
      this.match1 = {
        playerOne : +this.secondaryMatches[ s],
        playerTwo : +this.secondaryMatches[ s + 1],
        journeyId : +this.journeys[this.lastJourney + 1].id
      };
        // POST MATCHES
        this.matchesService.postMatch(this.match1)
```

```
        .subscribe( (match => {
          countMatches++;
          if ( countMatches === (this.secondaryMatches.length / 2)) {
             this.loadingService.hide();

this.alertService.show(this.translateService.instant('TENNIS.UPDATED MATCHES'));
          }
        }),
        ((error: Response) => {
          this.loadingService.hide();
          this.alertService.show(error.toString());
        }));
     }
   }

  showInformation() {
    this.show === true ? this.show = false : this.show = true;
  }

  showResults() {
    this.option === 'Manualmente' ? this.option = 'Aleatoriamente' : this.option =
'Manualmente';
  }

  gotoTournament() {
    const url = this.route.snapshot.url.join('/') + '/tournaments';
    this.router.navigate([url]);
  }

  gotoJourneys() {
    const url = this.route.snapshot.url.join('/') + '/journeys';
    this.router.navigate([url]);
  }

  gotoTeams() {
    const url = this.route.snapshot.url.join('/') + '/teams';
    this.router.navigate([url]);
  }

  deleteCompetition() {
    const dialogRef = this.dialog.open(DeleteCompetitionComponent, {
      data: { competition: this.competition, journeys: this.journeys }
    });
  }

}
```

```
<div *ngIf="competition" class="tennis-content">
  <div *ngIf="utilsService.role === 1 || utilsService.role === 0">
    <h5> {{ 'COMPETITIONS.TITLE' | translate }} > {{ competition.name }} </h5>
    <div *ngIf="finished">
    <div class="a-content">
     <a mat-raised-button class="tournament" (click)="gotoTournament()">
       <strong>{{ 'TENNIS.TOURNAMENT_TRACKING' | translate }}</strong>
     </a>
    </div>

    <div class="a-content">
        <a mat-raised-button class="journeys" (click)="gotoJourneys()">
          <strong>{{ 'COMPETITIONS.JOURNEYS' | translate }}</strong>
        </a>
   </div>

    <div class="a-content" *ngIf="competition.mode === 'Equipos'">
      <a mat-raised-button class="teams" (click)="gotoTeams()">
       <strong>{{ 'COMPETITIONS.TEAMS' | translate }}</strong>
      </a>
    </div>

    <div *ngIf="utilsService.role === 1" class="accordeon-journeys-info-results">

      <mat-accordion>

        <mat-expansion-panel class="option1">
          <mat-expansion-panel-header>
            <mat-panel-title>
```

```
              <strong  class="edit">{{ 'COMPETITIONS.MOD DATE' | translate }}</strong>
            </mat-panel-title>
            <mat-panel-description>
              {{ 'COMPETITIONS.MOD_DATE2' | translate }}
            </mat-panel-description>
          </mat-expansion-panel-header>

               <!-- Cuantas jornadas haya-->

                  <form [formGroup]="journeysFormGroup">
                      <div *ngIf="notCompletedJourneys.length === 0">
                          {{ 'COMPETITIONS.INTR RESULT4' | translate }} {{
'COMPETITIONS.NO MODIFY' | translate }}.
                      </div>
                      <div *ngIf="notCompletedJourneys.length !== 0">
                    <mat-form-field>
                        <mat-select #journeyId placeholder="{{
'COMPETITION CREATION.JOURNEY' | translate }}"
                             formControlName="id" required>
                          <mat-option *ngFor="let journey of notCompletedJourneys"
[value]="journey.id">
                            {{ journey.number }}
                          </mat-option>
                        </mat-select>
                    </mat-form-field>

                    <mat-form-field>
                      <input matInput [matDatepicker]="picker" formControlName="date"
                        placeholder="{{ 'COMPETITION CREATION.DATE' | translate }}"
required>
                      <mat-datepicker-toggle matSuffix [for]="picker"></mat-
datepicker-toggle>
                      <mat-datepicker #picker></mat-datepicker>
                    </mat-form-field>
                  </div>
                    <div *ngIf="notCompletedJourneys.length !== 0">
                      <button mat-raised-button class="edit-btn"
(click)="onSubmitJourney(journeysFormGroup.value)"
[disabled]="!journeysFormGroup.valid">
                          <b>{{ 'COMPETITIONS.MODIFY' | translate }}</b>
                      </button>
                    </div>

                  </form>
              </mat-expansion-panel>

              <mat-expansion-panel class="option2">
                <mat-expansion-panel-header>
                  <mat-panel-title>
                      <strong class="edit">{{ 'COMPETITIONS.MOD TEXT' | translate
}}</strong>
                  </mat-panel-title>
                  <mat-panel-description>
                    {{ 'COMPETITIONS.MOD_TEXT2' | translate }}
                  </mat-panel-description>
                </mat-expansion-panel-header>

                <form [formGroup]="informationFormGroup">
                <mat-form-field class="text-info">
                    <textarea matInput [value]="information"
                     formControlName="information" required></textarea>
                </mat-form-field>
                <div>
                    <button mat-raised-button class="edit-btn"
(click)="onSubmitInformation(informationFormGroup.value)"
[disabled]="!informationFormGroup.valid">
                        <b>{{ 'COMPETITIONS.MODIFY' | translate }}</b>
                    </button>
                </div>
                 </form>
              </mat-expansion-panel>

              <mat-expansion-panel class="option3">
                <mat-expansion-panel-header (click)="loadResultSection()">
                  <mat-panel-title>
                   <b>{{ 'COMPETITIONS.INTR_RESULT' | translate }}</b>
                  </mat-panel-title>
```

```
                  <mat-panel-description>
                   {{ 'COMPETITIONS.INTR_RESULT2' | translate }}
                  </mat-panel-description>
                </mat-expansion-panel-header>
                <div *ngIf="notCompletedJourneys.length !== 0">
                <div *ngIf=!matchesUploaded>
                  <p><b>¿{{ 'COMPETITIONS.QUESTION RESULTS' | translate }}
{{lastJourney + 1}}?</b></p>
                  <button [disabled]="option === 'Manualmente'" class="btn-rnd" mat-
raised-button (click)="showResults()">
                    <b>{{ 'COMPETITIONS.MANUALLY' | translate }}</b>
                  </button>
                  <button [disabled]="option === 'Aleatoriamente'" class="btn-rnd"
mat-raised-button (click)="showResults()">
                    <b>{{ 'COMPETITIONS.RANDOM' | translate }}</b>
                  </button>
                  <p>{{ 'COMPETITIONS.SELECTED' | translate }}: <b>{{option}}</b> </p>

                  <mat-card *ngIf="option === 'Manualmente'">
                   <div *ngIf="competition.mode === 'Individual'"><p><b>{{
'TENNIS.INTR_RESULT3' | translate }} {{lastJourney + 1}}:</b></p></div>
                   <div *ngIf="competition.mode === 'Equipos'"><p><b>{{
'TENNIS.INTR RESULT3 TEAM' | translate }} {{lastJourney + 1}}:</b></p></div>
                    <form [formGroup]="resultsFormGroup">
                    <div *ngIf="showMatchesPrimary">
                    <div *ngIf="showMatchesPrimary.length !== 0 ||
ghostsPrimary.length !== 0">
                     <p><u>{{ 'TENNIS.PRINCIPAL TOURNAMENT' | translate }}:</u></p>
                      <div class="form-group" formArrayName="results">
                        <div *ngFor="let match of
resultsFormGroup.controls.results.controls ;let i=index;">
                          <div class="panel-body" [formGroupName]="i">
                            <mat-radio-group  formControlName="winner" required>
                              <div>
                                <mat-radio-button class="match{{n}}" [value]="match"
*ngFor="let match of showMatchesPrimary[i];let n=index;">
                                  {{ match }}
                                </mat-radio-button>
                              </div>
                            </mat-radio-group>
                          </div>
                        </div>
                      </div>
                      </div>
                      <div *ngIf="ghostsPrimary">
                        <p class="info-ghosts" *ngIf="ghostsPrimary.length !== 0">
                            {{ 'TENNIS.NOTE PRIMARY' | translate }}
                        </p>
                      </div>
                    </div>
                      <div *ngIf="showMatchesSecondary">
                        <div *ngIf="showMatchesSecondary.length !== 0 ||
ghostsSecondary.length !== 0">
                         <p><u>{{ 'TENNIS.SECONDARY_TOURNAMENT' | translate }}:</u></p>
                        <div class="form-group" formArrayName="results2">
                            <div *ngFor="let match of
resultsFormGroup.controls.results2.controls ;let s=index;">
                             <div class="panel-body" [formGroupName]="s">
                               <mat-radio-group  formControlName="winner" required>
                                 <div>
                                   <mat-radio-button class="match{{n}}" [value]="match"
*ngFor="let match of showMatchesSecondary[s];let n=index;">
                                     {{ match }}
                                   </mat-radio-button>
                                 </div>
                               </mat-radio-group>
                             </div>
                            </div>
                          </div>
                        </div>
                        <div *ngIf="ghostsSecondary">
                            <p class="info-ghosts" *ngIf="ghostsSecondary.length !== 0">
                                {{ 'TENNIS.NOTE SECONDARY' | translate }}
                            </p>
                        </div>
                        </div>
                        <div>
```

```
                        <button mat-raised-button class="edit-btn"
(click)="onSubmitResults(resultsFormGroup.value)"
[disabled]="!resultsFormGroup.valid">
                          <b>{{ 'COMPETITIONS.INTR_RESULT1' | translate }}</b>
                        </button>
                      </div>
                    </form>
                  </mat-card >
                  <mat-card  *ngIf="option === 'Aleatoriamente'">
                      <p><b>{{ 'COMPETITIONS.RANDOM2' | translate }} {{lastJourney +
1}} {{ 'COMPETITIONS.RANDOM3' | translate }} {{ 'COMPETITIONS.INTR_RESULT1' |
translate }}:</b></p>
                      <button mat-raised-button class="edit-btn"
(click)="onSubmitResults()">
                          <b>{{ 'COMPETITIONS.INTR_RESULT1' | translate }}</b>
                      </button>
                  </mat-card >
              </div>
              <div *ngIf=matchesUploaded>
                  {{ 'COMPETITIONS.INTR_RESULT5' | translate }} {{lastJourney + 1}} {{
'COMPETITIONS.RESULTS_SUCCESS' | translate }}
              </div>
              </div>

              <div *ngIf="notCompletedJourneys.length === 0">
                {{ 'COMPETITIONS.INTR_RESULT4' | translate }}
              </div>
              </mat-expansion-panel>
      </mat-accordion>
      </div>

    <div class="show-info-content">
      <button *ngIf="!show"  mat-raised-button (click)="showInformation()">
          <mat-icon>info</mat-icon>
          <b>{{ 'COMPETITIONS.SHOW INFO' | translate }}</b>
      </button>
      <button *ngIf="show"  mat-raised-button (click)="showInformation()">
          <mat-icon>info_outline</mat-icon>
          <b>{{ 'COMPETITIONS.HIDE INFO' | translate }}</b>
      </button>
      <mat-card *ngIf="show" class="information">
          <mat-card-header>
            <mat-card-title><b>{{ 'COMPETITIONS.INFO_COMP' | translate }}: <span
class="competi-name">{{competition.name}}</span></b></mat-card-title>
            <mat-card-subtitle>{{ 'COMPETITIONS.TYPE' | translate }}:
{{competition.type}}, {{ 'COMPETITIONS.MODE' | translate }}:
{{competition.mode}}</mat-card-subtitle>
          </mat-card-header>
          <mat-card-content>
              {{competition.information}}
          </mat-card-content>
        </mat-card>
  </div>

  <div class="a-content" *ngIf="utilsService.role === 0">
    <a mat-raised-button>
      <strong>{{ 'COMPETITIONS.PLAY' | translate }}</strong>
    </a>
  </div>
  <div class="a-content" *ngIf="utilsService.role === 1">
    <button mat-raised-button class="delete" (click)="deleteCompetition()">
      <mat-icon>delete</mat-icon>
      <strong>{{ 'COMPETITIONS.DELETE' | translate }}</strong>
    </button>
  </div>
 </div>
    <div class="final-button">
        <a mat-button [routerLink]="'/competitions'">
          <mat-icon mat-list-icon>navigate before</mat-icon>
          {{ 'COMPETITIONS.RETURN2' | translate }}
        </a>
    </div>
  </div>
  </div>
```

9.7.10. Seguimiento del torneo de tenis

```
export class TournamentsComponent implements OnInit {

  public final = false;
  public finished = false;
  public tournamentCompleted = false;
  public winner: string;

  public competitionId: number;
  public competition: Competition;

  public journeys: Journey[];
  public matchesJourneys: Match[][];
  public participants: any[];

  public lastJourney: number;
  public participantsPrimary: String[];
  public participantsSecondary: String[];
  public participantsEliminated: String[];
  public ghostIndex: number;

  constructor(public alertService: AlertService,
    public utilsService: UtilsService,
    public loadingService: LoadingService,
    public competitionService: CompetitionService,
    public journeyService: JourneyService,
    public matchesService: MatchesService,
    public teamService: TeamService,
    private route: ActivatedRoute) {
      this.utilsService.currentUser                                                    =
Login.toObject(localStorage.getItem(AppConfig.LS USER));
      this.utilsService.role = Number(localStorage.getItem(AppConfig.LS ROLE));
     }

  ngOnInit() {
    if   (this.utilsService.role   ===   Role.TEACHER   ||   this.utilsService.role   ===
Role.STUDENT) {
      this.loadingService.show();
      this.competitionId = +this.route.snapshot.paramMap.get('id');
      this.getSelectedCompetition();
    }
  }
  /** This method returnsthe current competition and calls the getMatches method */
  private getSelectedCompetition(): void {
    this.competitionService.getCompetition(this.competitionId).subscribe(
      ((competition: Competition) => {
        this.competition = competition;
        this.getJourneys();
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
  }
  /**
   * This method returns the journeys of the current competition
   * and calls the getMatches method
   */
  private getJourneys(): void {
    this.journeyService.getJourneysCompetition(this.competitionId).subscribe(
      ((journeys: Array<Journey>) => {
        this.journeys = journeys;
        this.journeys.sort(function (a, b) { return (a.number - b.number); });
                    // tslint:disable-next-line:no-console
                    console.log(this.journeys);
        this.getMatches();
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
  }
  /**
   * This method returns the matches of each journey
```

```
   * and calls the getParticipants method
   */
  private getMatches(): void {
    this.matchesJourneys = [];
    let journeysCompleted = 0;
    for (let  n = 0;  n < this.journeys.length;  n++) {
      this.journeyService.getMatchesJourneyDetails(this.journeys[ n].id,
this.competition).subscribe(
      ((matches: Array<Match>) => {
        this.matchesJourneys[_n] = [];
        for (let _m = 0; _m < matches.length; _m++) {
          this.matchesJourneys[ n][ m] = new Match();
          this.matchesJourneys[ n][ m] = matches[ m];
        }
        journeysCompleted++;
        if (this.matchesJourneys[_n][0].winner === 0) { this.lastJourney = _n; }
        if ( journeysCompleted === this.lastJourney + 1 || this.matchesJourneys.length
=== this.journeys.length) {
          if   (   this.lastJourney   ===   undefined   )   {   this.lastJourney   =
this.journeys.length - 1; }
          // tslint:disable-next-line:no-console
          console.log(this.matchesJourneys);
          this.getParticipants();
         }
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
    }
  }
  /**
   * This method returns the participants of the current competition
   * and calls the getTournamentStatus method
   */
  private getParticipants(): void {
    this.participants = [];
    if (this.competition.mode === 'Individual') {
      this.competitionService.getStudentsCompetition(this.competition.id)
      .subscribe(( (students: Array<Student>) => {
        for (let  s = 0;  s < students.length;  s++) {
          this.participants[_s] = {
            id: +students[_s].id,
            name: students[_s].name.concat(' ', students[_s].surname)
          };
        }
        this.getTournamentStatus();
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
      } else {
      this.teamService.getTeamsCompetition(this.competitionId)
      .subscribe(( (teams: Array<Team>) => {
        for (let  t = 0;  t < teams.length;  t++) {
          this.participants[ t] = {
            id: +teams[ t].id,
            name: teams[_t].name
          };
        }
        this.getTournamentStatus();
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
      }
  }
  /**
   * This method divides the participants between the main tournament,
   *  the secondary tournament and the eliminated ones
   */
  private getTournamentStatus(): void {

   this.participantsPrimary = [];
   this.participantsSecondary = [];
```

```
    for (let _m = 0; _m < this.matchesJourneys[this.lastJourney].length; _m++) {
     if ( this.lastJourney === 0 ) {

this.participantsPrimary.push(this.matchesJourneys[this.lastJourney][_m].namePlayerOne
);

this.participantsPrimary.push(this.matchesJourneys[this.lastJourney][_m].namePlayerTwo
);
     } else if ( (this.lastJourney + 1) % 2 === 0 && this.lastJourney + 1 !==
this.journeys.length ) {
       if (  m < this.matchesJourneys[this.lastJourney].length / 2 ) {

this.participantsPrimary.push(this.matchesJourneys[this.lastJourney][_m].namePlayerOne
);

this.participantsPrimary.push(this.matchesJourneys[this.lastJourney][_m].namePlayerTwo
);
       } else {

this.participantsSecondary.push(this.matchesJourneys[this.lastJourney][_m].namePlayerO
ne);

this.participantsSecondary.push(this.matchesJourneys[this.lastJourney][ m].namePlayerT
wo);
       }
     } else if ( (this.lastJourney + 1) % 2 !== 0 ) {
      this.matchesJourneys[this.lastJourney         -         1][ m].winner         ===
this.matchesJourneys[this.lastJourney - 1][ m].playerOne ?
      this.participantsPrimary.push(this.matchesJourneys[this.lastJourney             -
1][_m].namePlayerOne) :
      this.participantsPrimary.push(this.matchesJourneys[this.lastJourney             -
1][ m].namePlayerTwo);

this.participantsSecondary.push(this.matchesJourneys[this.lastJourney][ m].namePlayerO
ne);

this.participantsSecondary.push(this.matchesJourneys[this.lastJourney][_m].namePlayerT
wo);
     } else if ( this.lastJourney + 1 === this.journeys.length ) {
      this.final = true;
      if ( this.matchesJourneys[this.lastJourney][0].winner !== 0) {
        this.tournamentCompleted = true;
        this.matchesJourneys[this.lastJourney][0].winner                              ===
this.matchesJourneys[this.lastJourney][0].playerOne ?
        this.winner = this.matchesJourneys[this.lastJourney][0].namePlayerOne :
        this.winner = this.matchesJourneys[this.lastJourney][0].namePlayerTwo;
      }

this.participantsPrimary.push(this.matchesJourneys[this.lastJourney][ m].namePlayerOne
);

this.participantsPrimary.push(this.matchesJourneys[this.lastJourney][_m].namePlayerTwo
);
     }
    }

    // Deleting ghosts to show
    this.ghostIndex = 0;
    while ( this.ghostIndex < this.participantsPrimary.length) {
      if (this.participantsPrimary[this.ghostIndex] === 'Ghost') {
        this.participantsPrimary.splice(this.ghostIndex, 1);
        this.ghostIndex = 0;
      } else { this.ghostIndex++; }
    }
    this.ghostIndex = 0;
    while ( this.ghostIndex < this.participantsSecondary.length) {
      if (this.participantsSecondary[this.ghostIndex] === 'Ghost') {
        this.participantsSecondary.splice(this.ghostIndex, 1);
        this.ghostIndex = 0;
      } else { this.ghostIndex++; }
    }

    // Adding eliminated participants
    this.participantsEliminated = [];
    for (let _d = 0; _d < this.participants.length; _d++) {
      let count = 0;
```

```
      for (let  p = 0;  p < this.participantsPrimary.length;  p++) {
        if ( this.participants[_d].name === this.participantsPrimary[_p] ) {
          count = 1;
        }
      }
      if (count === 0) { this.participantsEliminated.push(this.participants[ d].name);
}
    }

    let _q = 0;
    while (_q < this.participantsEliminated.length) {
      let count = 0;
      for (let  p = 0;  p < this.participantsSecondary.length;  p++) {
        if ( this.participantsEliminated[_q] === this.participantsSecondary[_p] ) {
          count = 1;
        }
      }
      if (count === 1) {
       this.participantsEliminated.splice( q, 1);
       _q = 0;
      } else { _q++; }
    }
    this.loadingService.hide();
    this.finished = true;
  }
}
```

```
<div *ngIf="utilsService.role === 1 || utilsService.role === 0" class="tournaments-
content">
  <h5 *ngIf="competition"> {{ 'COMPETITIONS.TITLE' | translate }} > {{
competition.name }} >
      {{ 'TENNIS.TOURNAMENT TRACKING' | translate }} </h5>

  <div *ngIf="finished">
    <h3 class="title">{{ 'TENNIS.TOURNAMENT_TRACKING' | translate }}: {{
competition.name }}</h3>
    <mat-card *ngIf="winner" class="winner-card">
     <b>{{ 'TOURNAMENTS.FINISHED' | translate }}.</b>
     <p class="winner"><b>{{ 'COMPETITIONS.WINNER' | translate }}: {{winner}}</b></p>
    </mat-card>
    <h4 *ngIf="!winner">{{ 'TOURNAMENTS.PARTICIPANTS' | translate }} ({{
'COMPETITIONS.JOURNEY_NUMBER' | translate }} {{lastJourney + 1}}):</h4>
    <div *ngIf="!final">
    <mat-card class="tournaments-card">
      <mat-card-content>
        <mat-list>
          <h3 class="subtitle">{{ 'TOURNAMENTS.PRINCIPAL' | translate }}</h3>
          <mat-list-item *ngFor="let primary of participantsPrimary">
            <mat-icon mat-list-icon *ngIf="competition.mode ===
'Individual'">person</mat-icon>
            <mat-icon mat-list-icon *ngIf="competition.mode === 'Equipos'">group</mat-
icon>
            <p>{{primary}}</p>
          </mat-list-item>
          <mat-divider></mat-divider>
          <h3 class="subtitle">{{ 'TOURNAMENTS.SECONDARY' | translate }}</h3>
          <mat-list-item *ngFor="let secondary of participantsSecondary">
            <mat-icon mat-list-icon *ngIf="competition.mode ===
'Individual'">person</mat-icon>
            <mat-icon mat-list-icon *ngIf="competition.mode === 'Equipos'">group</mat-
icon>
            <p>{{secondary}}</p>
          </mat-list-item>
          <mat-list-item *ngIf="participantsSecondary.length === 0">{{
'TOURNAMENTS.NO PLAYER SECONDARY' | translate }}.</mat-list-item>
        </mat-list>
      </mat-card-content>
    </mat-card>
  </div>
  <div *ngIf="final">
   <h4 class="final">{{ 'TOURNAMENTS.FINAL' | translate }}:</h4>
     <mat-card class="eliminated-card">
       <mat-card-content>
         <mat-list>
           <h3 class="subtitle">{{ 'TOURNAMENTS.FINALISTS' | translate }}</h3>
           <mat-list-item *ngFor="let final of participantsPrimary">
```

```
            <mat-icon mat-list-icon *ngIf="competition.mode ===
'Individual'">person</mat-icon>
            <mat-icon mat-list-icon *ngIf="competition.mode ===
'Equipos'">group</mat-icon>
            <p>{{final}}</p>
          </mat-list-item>
        </mat-list>
      </mat-card-content>
    </mat-card>
  </div>
  <div>
      <h4>{{ 'TOURNAMENTS.PLAYERS ELIMINATED' | translate }}:</h4>
      <mat-card class="eliminated-card">
        <mat-card-content>
          <mat-list>
            <h3 class="subtitle">{{ 'TOURNAMENTS.ELIMINATED' | translate }}</h3>
            <mat-list-item *ngFor="let eliminated of participantsEliminated">
              <mat-icon mat-list-icon *ngIf="competition.mode ===
'Individual'">person</mat-icon>
              <mat-icon mat-list-icon *ngIf="competition.mode ===
'Equipos'">group</mat-icon>
              <p>{{eliminated}}</p>
            </mat-list-item>
            <mat-list-item *ngIf="participantsEliminated.length === 0">{{
'TOURNAMENTS.NO_PLAYER_ELIMINATED' | translate }}.</mat-list-item>
          </mat-list>
        </mat-card-content>
      </mat-card>
    </div>
  </div>
  <div class="final-buttons">
      <a mat-button [routerLink]="['/competition/tennis', competitionId]">
        <mat-icon mat-list-icon>navigate before</mat-icon>
        {{ 'COMPETITIONS.RETURN' | translate }}
      </a>
  </div>
</div>
```

9.7.11. Calendario del torneo de tenis

```
export class JourneysTennisComponent implements OnInit {

  public show: boolean;
  public results: boolean;
  public competitionId: string;
  public competition: Competition;

  public journeys = new Array<Journey>();
  public numJourneys: number;
  public dates: String[];
  public datesNoMatches: any[];

  public matchesJourneys: Match[][];
  public matchesPrincipal: Match[][];
  public matchesSecondary: Match[][];
  public lastJourney: number;
  public tournaments: String[][];

  constructor(public alertService: AlertService,
    public utilsService: UtilsService,
    public translateService: TranslateService,
    public loadingService: LoadingService,
    public competitionService: CompetitionService,
    public journeyService: JourneyService,
    public teamService: TeamService,
    public datePipe: DatePipe,
    private route: ActivatedRoute) {
      this.utilsService.currentUser                                                    =
Login.toObject(localStorage.getItem(AppConfig.LS USER));
      this.utilsService.role = Number(localStorage.getItem(AppConfig.LS ROLE));
      this.show = false;
      this.results = false;
     }
```

```
  ngOnInit() {
    if ( this.utilsService.role === Role.TEACHER || this.utilsService.role ===
Role.STUDENT ) {
      this.loadingService.show();
      this.competitionId = this.route.snapshot.paramMap.get('id');
      this.getSelectedCompetition();
    }
  }
  /**
   * This method returns the selected competition by id
   * and calls the getJourneys method
   */
  private getSelectedCompetition(): void {
    this.competitionService.getCompetition(this.competitionId).subscribe(
      ((competition: Competition) => {
        this.competition = competition;
        this.getJourneys();
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
  }
  /**
   * This method returns the journeys of the current competition
   * and calls the getMatches method
   */
  private getJourneys(): void {
    this.journeyService.getJourneysCompetition(this.competitionId).subscribe(
      ((journeys: Array<Journey>) => {
        this.journeys = journeys;
        this.journeys.sort(function (a, b) {
          return (a.number - b.number);
        });
        this.numJourneys = this.journeys.length;
        this.getMatches();
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
  }
  /**
   * This method returns the matches of each journey
   * and calls the getDatesAndResults method
   */
  private getMatches(): void {
    let countJourneys = 0;
    this.matchesJourneys = [];
    for (let j = 0; j < this.journeys.length; j++) {
      this.journeyService.getMatchesJourneyDetails(this.journeys[ j].id,
this.competition).subscribe(
      ((matches: Array<Match>) => {
        this.matchesJourneys[_j] = [];
        for (let m = 0; m < matches.length; m++) {
          this.matchesJourneys[ j][ m] = new Match();
          this.matchesJourneys[ j][ m] = matches[ m];
        }
        if (this.matchesJourneys[_j][0].winner === 0) { this.lastJourney = _j; }
        countJourneys++;
        if ( countJourneys === this.lastJourney + 1 || countJourneys ===
this.numJourneys) {
          if ( this.lastJourney === undefined ) { this.lastJourney = this.numJourneys
- 1; }
          this.getDatesAndResults();
        }
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
    }
  }
  /**
   * This method make the date and results of each journey
   * and the rest of content to show in the journeys-tennis page
```

```
   */
  private getDatesAndResults(): void {
    this.dates = [];
    this.datesNoMatches = [];
    this.tournaments = [];
    // DATES AND TOURNAMENTS
    for (let  j = 0;  j < this.journeys.length;  j++) {
      if ( _j <= this.lastJourney) {
      this.journeys[_j].date === null ?
        this.dates[_j]  =  this.translateService.instant('TOURNAMENTS.NOT_ESTABLISHED')
:
        this.dates[ j]   =   this.datePipe.transform(this.journeys[ j].date,   'dd-MM-
yyyy');
      } else {
        this.journeys[_j].date === null ?
        this.datesNoMatches.push({date:
this.translateService.instant('TOURNAMENTS.NOT_ESTABLISHED'), number: _j + 1}) :
        this.datesNoMatches.push({date:
this.datePipe.transform(this.journeys[ j].date, 'dd-MM-yyyy'), number:  j + 1});
        if ((_j + 1) % 2 === 0 && _j !== this.journeys.length - 1) { // si es par y no
es el final participa en ambos
          this.tournaments.push([
            this.translateService.instant('TOURNAMENTS.PRINCIPAL')    +    ':    '    +
this.translateService.instant('TOURNAMENTS.PARTICIPATES'),
            this.translateService.instant('TOURNAMENTS.SECONDARY')    +    ':    '    +
this.translateService.instant('TOURNAMENTS.PARTICIPATES')]);
        } else if (( j + 1) % 2 !== 0) {
          this.tournaments.push([
            this.translateService.instant('TOURNAMENTS.PRINCIPAL')    +    ':    '    +
this.translateService.instant('TOURNAMENTS.DONT_PARTICIPATES'),
            this.translateService.instant('TOURNAMENTS.SECONDARY')    +    ':    '    +
this.translateService.instant('TOURNAMENTS.PARTICIPATES')]);
        } else if (  j === this.journeys.length - 1) {
          this.tournaments.push([this.translateService.instant('TOURNAMENTS.FINAL')]);
        }
      }
    }
    // INTRODUCE THE RESULT AND TO SEPARATE IN PRINCIPAL AND SECONDARY MATCHES
    this.matchesPrincipal = [];
    this.matchesSecondary = [];
    for (let  j = 0;  j < this.matchesJourneys.length;  j++) {
      this.matchesPrincipal[_j] = [];
      this.matchesSecondary[_j] = [];
      for (let _m = 0; _m < this.matchesJourneys[_j].length; _m++) {
        if ( this.matchesJourneys[ j][ m].winner === 0 ) {
          this.matchesJourneys[ j][ m].result = '-';
        } else {
          this.matchesJourneys[_j][_m].winner                                          ===
this.matchesJourneys[_j][_m].playerOne ?
          this.matchesJourneys[ j][ m].result                                           =
this.matchesJourneys[ j][ m].namePlayerOne :
          this.matchesJourneys[ j][ m].result                                           =
this.matchesJourneys[_j][_m].namePlayerTwo;
        }
        if ((_j + 1) % 2 === 0) { // si es par participa en ambos
           m < this.matchesJourneys[ j].length / 2 ?
          this.matchesPrincipal[ j].push(this.matchesJourneys[ j][ m]) :
          this.matchesSecondary[ j].push(this.matchesJourneys[ j][ m]);
        } else if ((_j + 1) % 2 !== 0 && _j !== 0) { // si es impar participa solo en
el secundario excepto en el primer partido
          this.matchesSecondary[_j].push(this.matchesJourneys[_j][_m]);
        } else if (  j === 0) {
          this.matchesPrincipal[ j].push(this.matchesJourneys[ j][ m]);
        }
      }
    }
    this.results = true;
    this.loadingService.hide();
  }

  private showMore() {
    this.show === true ? this.show = false : this.show = true;
  }
}
```

```
<div *ngIf="utilsService.role === 1 || utilsService.role === 0" class="journeys-
content">
    <h5 *ngIf="competition"> {{ 'COMPETITIONS.TITLE' | translate }} > {{
competition.name }} >
          {{ 'COMPETITIONS.CALENDAR' | translate }} </h5>

    <div *ngIf="results">
    <h4>{{ 'COMPETITIONS.CALENDAR_OF2' | translate }} {{ 'TENNIS.TITLE' | translate }}:
{{ competition.name }}</h4>
        <mat-card class="journey-card" *ngFor="let date of dates; let i=index;">
          <mat-card-content>
              <p><u>{{ 'COMPETITIONS.JOURNEY' | translate }} {{i + 1}}</u></p>
              <p    class="date"><i>{{    'COMPETITIONS.DATE'    |    translate    }}:
{{date}}</i></p>
           <table *ngIf="matchesPrincipal[i].length !== 0">
              <caption>
               <p *ngIf="i !== numJourneys - 1">{{ 'TOURNAMENTS.PRINCIPAL' | translate
}}</p>
               <p *ngIf="i === numJourneys - 1">{{ 'TOURNAMENTS.FINAL' | translate
}}</p>
              </caption>
              <thead>
                <tr class="tableHeader">
                  <th *ngIf="competition.mode === 'Individual'" class="player">{{
'COMPETITIONS.PLAYER' | translate }} 1</th>
                  <th *ngIf="competition.mode === 'Individual'" class="player">{{
'COMPETITIONS.PLAYER' | translate }} 2</th>
                  <th *ngIf="competition.mode === 'Equipos'" class="player">{{
'COMPETITIONS.TEAM' | translate }} 1</th>
                  <th *ngIf="competition.mode === 'Equipos'" class="player">{{
'COMPETITIONS.TEAM' | translate }} 2</th>
                  <th class="result">{{ 'COMPETITIONS.WINNER' | translate }}</th>
                </tr>
              </thead>
              <tbody>
                <tr *ngFor="let match of matchesPrincipal[i]">
                  <td class="player">{{match.namePlayerOne}}</td>
                  <td class="player">{{match.namePlayerTwo}}</td>
                  <td class="result">{{match.result}}</td>
                </tr>
              </tbody>
            </table>
            <div class="thead-tbody" *ngIf="matchesSecondary[i].length !== 0">
            <p>{{ 'TOURNAMENTS.SECONDARY' | translate }}</p>
            <table>
              <thead>
                <tr class="tableHeader">
                  <th *ngIf="competition.mode === 'Individual'" class="player">{{
'COMPETITIONS.PLAYER' | translate }} 1</th>
                  <th *ngIf="competition.mode === 'Individual'" class="player">{{
'COMPETITIONS.PLAYER' | translate }} 2</th>
                  <th *ngIf="competition.mode === 'Equipos'" class="player">{{
'COMPETITIONS.TEAM' | translate }} 1</th>
                  <th *ngIf="competition.mode === 'Equipos'" class="player">{{
'COMPETITIONS.TEAM' | translate }} 2</th>
                  <th class="result">{{ 'COMPETITIONS.WINNER' | translate }}</th>
                </tr>
              </thead>
              <tbody>
                <tr *ngFor="let match2 of matchesSecondary[i]">
                  <td class="player">{{match2.namePlayerOne}}</td>
                  <td class="player">{{match2.namePlayerTwo}}</td>
                  <td class="result">{{match2.result}}</td>
                </tr>
              </tbody>
            </table>
          </div>
          </mat-card-content>
        </mat-card>
    </div>
    <div class="foot">
        <mat-card>
      * {{ 'TENNIS.GHOST1' | translate }}
      {{ 'TENNIS.GHOST2' | translate }}.
      </mat-card>
    </div>
    <div *ngIf="datesNoMatches">
```

```
      <div *ngIf="datesNoMatches.length !==0">
        <h6   *ngIf="!show">{{   'TENNIS.MORE_JOURNEY1'   |   translate   }}.   {{
'TENNIS.MORE_JOURNEY2' | translate }}:</h6>
        <h6 *ngIf="show">{{ 'TENNIS.LESS_JOURNEY' | translate }}:</h6>
        <button   mat-fab   color="warn"   *ngIf="!show"   (click)="showMore()"><mat-
icon>add</mat-icon></button>
        <button   mat-fab   color="warn"   *ngIf="show"   (click)="showMore()"><mat-
icon>remove</mat-icon></button>
        <div *ngIf="show" class="cardsNoMatch-div">
        <mat-card class="dateNoMatch-card" *ngFor="let dateNoMatch of datesNoMatches;
let i=index;">
            <mat-card-header>
                <mat-card-title><p><u>{{   'COMPETITIONS.JOURNEY'   |   translate   }}
{{dateNoMatch.number}}</u></p></mat-card-title>
                <mat-card-subtitle>{{   'COMPETITIONS.DATE'   |   translate   }}:
{{dateNoMatch.date}}</mat-card-subtitle>
              </mat-card-header>
            <mat-card-content>
              <mat-divider></mat-divider>
              <mat-list role="list">
                 <mat-list-item   role="listitem"   *ngFor="let   tournament   of
tournaments[i]">- {{tournament}}</mat-list-item>
              </mat-list>
            </mat-card-content>
          </mat-card>
    </div>
  </div>
  </div>
  <div class="final-buttons">
      <a mat-button [routerLink]="['/competition/tennis', competitionId]">
        <mat-icon mat-list-icon>navigate_before</mat-icon>
        {{ 'COMPETITIONS.RETURN' | translate }}
      </a>
  </div>
  </div>
```

9.7.12. Equipos de la liga y del torneo de tenis (compartido)

Componente que se muestra los participantes de cada equipo

```
export class TeamsComponent implements OnInit {

  public competitionId: number;
  public competition: Competition;
  public teams: Team[];
  public allStudents: Student[][];

  constructor(public alertService: AlertService,
    public utilsService: UtilsService,
    public loadingService: LoadingService,
    public competitionService: CompetitionService,
    public teamService: TeamService,
    private route: ActivatedRoute) {
      this.utilsService.currentUser                                                =
Login.toObject(localStorage.getItem(AppConfig.LS_USER));
      this.utilsService.role = Number(localStorage.getItem(AppConfig.LS ROLE));
     }

  ngOnInit() {
    if   (this.utilsService.role   ===   Role.TEACHER   ||   this.utilsService.role   ===
Role.STUDENT) {
    this.loadingService.show();
    this.competitionId = +this.route.snapshot.paramMap.get('id');
    this.getSelectedCompetition();
    }
  }

  private getSelectedCompetition(): void {
    this.competitionService.getCompetition(this.competitionId).subscribe(
      ((competition: Competition) => {
        this.competition = competition;
        this.getTeams();
```

```
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
  }

  private getTeams(): void {
    this.teamService.getTeamsCompetition(this.competitionId)
    .subscribe((teams => {
        this.teams = teams,
        this.getStudents();
      }),
      ((error: Response) => {
        this.loadingService.hide();
        this.alertService.show(error.toString());
      }));
  }

  private getStudents(): void {
    let countTeams = 0;
    this.allStudents = [];
    for (let  t = 0;  t < this.teams.length;  t++) {
    this.teamService.getStudentsTeam(+this.teams[ t].id)
    .subscribe(students => {
      this.allStudents[_t] = students;
      this.teams[ t].numPlayers = students.length;
      countTeams = countTeams + 1;
      if ( countTeams === this.teams.length) { this.loadingService.hide(); }
    });
    }
  }

}
```

```
<div *ngIf="utilsService.role === 1 || utilsService.role === 0" class="teams-content">
  <div *ngIf="competition">
  <h5> {{ 'COMPETITIONS.TITLE' | translate }} > {{competition.name}} >
    {{ 'COMPETITIONS.TEAMS' | translate }} </h5>
  <h3>{{ 'COMPETITIONS.TEAMS' | translate }}</h3>
    <mat-card class="team-card" *ngFor="let team of teams; let i=index;">
        <mat-card-header>
          <mat-card-title><strong>{{ 'COMPETITIONS.TEAM' | translate }} {{i + 1}}:
{{team.name | uppercase}}</strong></mat-card-title>
          <mat-card-subtitle>{{team.numPlayers}} {{ 'COMPETITIONS.PLAYERS' | translate
}}</mat-card-subtitle>
        </mat-card-header>
        <mat-divider></mat-divider>
        <mat-card-content>
            <mat-list class="team-list" role="list" *ngFor="let student of
allStudents[i]">
                <mat-list-item class="team-list-item" role="listitem">
                    <mat-icon mat-list-icon>person</mat-icon>
                  {{student.name}} {{student.surname}}
                </mat-list-item>
            </mat-list>
        </mat-card-content>
      </mat-card>

      <div class="final-buttons">
          <a mat-button *ngIf="competition.type === 'Liga'"
[routerLink]="['/competition/league', competitionId]">
            <mat-icon mat-list-icon>navigate_before</mat-icon>
            {{ 'TEAMS.RETURN' | translate }}
          </a>
          <a mat-button *ngIf="competition.type === 'Tenis'"
[routerLink]="['/competition/tennis', competitionId]">
            <mat-icon mat-list-icon>navigate_before</mat-icon>
            {{ 'TEAMS.RETURN' | translate }}
          </a>
      </div>
    </div>
</div>
```

9.8. Servicios del dashboard

9.8.1. Competition.service.ts

```
export class CompetitionService {

  constructor(
    public http: Http,
    public utilsService: UtilsService,
    public groupService: GroupService,
    public gradeService: GradeService,
    public matterService: MatterService) { }

  /**
  * This method returns the list of competitions of the current
  * user groups with the group (grade and matter)
  * @return {Array<Competition>} returns the list of competitions
  */

  public getMyCompetitionsByGroup(group: Group): Observable<Array<Competition>> {
    const ret: Array<Competition> = new Array<Competition>();

    return Observable.create(observer => {
      this.getCompetitionsByGroup(group.id).subscribe(competitions => {
        competitions.forEach(competition => {
          competition.grade = group.grade;
          competition.matter = group.matter;
          ret.push(competition);
          if (ret.length === competitions.length) {
            observer.next(ret);
            observer.complete();
          }
        });
      }, error => observer.error(error));
    });
  }

  /**
  * This method returns the list of competitions by group
  * @return {Array<Competition>} returns the list of competitions
  */

  private getCompetitionsByGroup(groupId: string): Observable<Array<Competition>> {

    const options: RequestOptions = new RequestOptions({
      headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });

    const url: string = AppConfig.GROUP URL + '/' + groupId +
AppConfig.COMPETITIONS URL;

    return this.http.get(url, options)
      .map((response: Response, index: number) =>
Competition.toObjectArray(response.json()))
      .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

  /**
  * This method returns the list of competitions of
  * the current user logged into the application with
  * the group (grade and matter)
  * @return {Array<Competition>} returns the list of competitions
  */

  public getMyCompetitions(): Observable<Array<Competition>> {

    const ret: Array<Competition> = new Array<Competition>();

    return Observable.create(observer => {
     this.getCompetitions().subscribe(competitions => {
      competitions.forEach(competition => {
        this.groupService.getGroup(competition.groupId).subscribe(
         group => {
```

```
          this.gradeService.getGrade(group.gradeId).subscribe(
           grade => {
            competition.grade = grade;
            this.matterService.getMatter(group.matterId).subscribe(
             matter => {
              competition.matter = matter;
              ret.push(competition);
              if (ret.length === competitions.length) {
                observer.next(ret);
                observer.complete();
              }
             }, error => observer.error(error));
           }, error => observer.error(error));
         }, error => observer.error(error));
       });
     }, error => observer.error(error));
    });
  }

  /**
  * This method returns the list of competitions of
  * the current user logged into the application
  * @return {Array<Competition>} returns the list of competitions
  */

  private getCompetitions(): Observable<Array<Competition>> {

    const options: RequestOptions = new RequestOptions({
      headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });

    const url: string = this.utilsService.getMyUrl() + AppConfig.COMPETITIONS URL;

    return this.http.get(url, options)
      .map((response: Response, index: number) =>
Competition.toObjectArray(response.json()))
      .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

  /**
  * This method returns the competition by its id
  * @return {Observable<Competition>} returns the competition
  */
  public getCompetition(id: number | string): Observable<Competition> {

    const options: RequestOptions = new RequestOptions({
      headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });

    return this.http.get(AppConfig.COMPETITION URL + '/' + id, options)
      .map((response: Response, index: number) =>
Competition.toObject(response.json()))
      .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

  /**
  * POST: add a new competition to the database
  * @return {Observable<Competition>} returns the competition
  */
  public postCompetition (competition: Competition): Observable<Competition> {

    const options: RequestOptions = new RequestOptions({
      headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });

    return this.http.post(AppConfig.COMPETITION URL, competition, options)
      .map((response: Response, index: number) =>
Competition.toObject(response.json()))
      .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

  /**
  * PUT: add new information to the competition
  * @return {Observable<Competition>} returns the competition
```

```
  */
  public putInformation (information: string, id: string | number):
Observable<Competition> {

    const options: RequestOptions = new RequestOptions({
      headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });

    return this.http.put(AppConfig.COMPETITION_URL + '/' + id, information, options)
      .map((response: Response, index: number) =>
Competition.toObject(response.json()))
      .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

  /**
  * PUT: made the relation between a student and a competition
  * @return {Observable<Response>}
  */
  public relCompetitionStudent (competitionId: string | number, studentId: string |
number): Observable<Response> {
    const options: RequestOptions = new RequestOptions({
     headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });
    return this.http.put(AppConfig.COMPETITION_URL + '/' + competitionId +
AppConfig.STUDENTS URL
      + AppConfig.REL URL + '/' + studentId, Response , options)
      .map((response: Response) => response.json())
      .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

  /**
  * GET: get students of a competition
  * @return {Observable<Competition>} returns the list of students
  */
  public getStudentsCompetition(competitionId: string): Observable<Array<Student>> {

    const options: RequestOptions = new RequestOptions({
     headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });
    const url: string = AppConfig.COMPETITION_URL + '/' + competitionId +
AppConfig.STUDENTS_URL;

    return this.http.get(url, options)
      .map((response: Response, index: number) =>
Student.toObjectArray(response.json()))
      .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

  /** DELETE: delete the competition from the server */
  public deleteCompetition (competitionId: number): Observable<{}> {
    const options: RequestOptions = new RequestOptions({
      headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });
    const url = AppConfig.COMPETITION URL + '/' + competitionId;
    return this.http.delete(url, options)
      .map((response: Response) => response.json())
      .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

  /** DELETE: delete the journeys of one competition from the server */
  public deleteJourneysCompetition (competitionId: number): Observable<{}> {
    const options: RequestOptions = new RequestOptions({
      headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });
    const url = AppConfig.COMPETITION_URL + '/' + competitionId +
AppConfig.JOURNEYS_URL;
    return this.http.delete(url, options)
      .map((response: Response) => response.json())
      .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

  /** DELETE: delete the matches of one journey from the server */
```

```
  public deleteMatchesCompetition (journeyId: number): Observable<{}> {
    const options: RequestOptions = new RequestOptions({
      headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });
    const url = AppConfig.JOURNEY URL + '/' + journeyId + AppConfig.MATCHES URL;
    return this.http.delete(url, options)
      .map((response: Response) => response.json())
      .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

  /** DELETE: delete the partitipants of one competition from the server */
  public deleteParticipantsCompetition (competition: Competition): Observable<{}> {
    const options: RequestOptions = new RequestOptions({
      headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });

    if (competition.mode === 'Individual') {
      return this.http.delete(AppConfig.COMPETITION_URL + '/' + competition.id +
AppConfig.STUDENTS_URL, options)
       .map((response: Response) => response.json())
       .catch((error: Response) => this.utilsService.handleAPIError(error));
    } else {
      return this.http.delete(AppConfig.COMPETITION_URL + '/' + competition.id +
AppConfig.TEAMS_URL, options)
       .map((response: Response) => response.json())
       .catch((error: Response) => this.utilsService.handleAPIError(error));
    }
  }

}
```

9.8.2. Journey.service.ts

```
export class JourneyService {

  constructor(
    public http: Http,
    public utilsService: UtilsService,
    public competitionService: CompetitionService,
    public teamService: TeamService) { }

  /**
  * POST: add a new journey
  * @return {Observable<Journey>} returns the journey
  */
   public postJourney (journey: Journey): Observable<Journey> {

     const options: RequestOptions = new RequestOptions({
        headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
      });

      return this.http.post(AppConfig.JOURNEY_URL, journey, options)
      .map((response: Response, index: number) => Journey.toObject(response.json()))
      .catch((error: Response) => this.utilsService.handleAPIError(error));
   }

  /**
  * PUT: modify a journey
  * @return {Observable<Journey>} returns the modified journey
  */
   public putJourney (value): Observable<Journey> {

    const options: RequestOptions = new RequestOptions({
     headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
     });

       return this.http.put(AppConfig.JOURNEY_URL + '/' + value.id, value , options)
       .map((response: Response, index: number) => Journey.toObject(response.json()))
       .catch((error: Response) => this.utilsService.handleAPIError(error));
```

```
  }

  /**
  * Get: return the list of journeys of a competition
  * @return {Observable<Array<Journey>>} returns the list of journeys
  */
  public getJourneysCompetition(idCompetition: string | number):
Observable<Array<Journey>> {

    const options: RequestOptions = new RequestOptions({
      headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });

    const url: string = AppConfig.COMPETITION_URL + '/' + idCompetition +
AppConfig.JOURNEYS_URL;

    return this.http.get(url, options)
      .map((response: Response, index: number) =>
Journey.toObjectArray(response.json()))
      .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

  /**
  * Get: return the list of matches of a journey
  * @return {Observable<Array<Match>>} returns the list of matches
  */
  private getMatchesJourney(idJourney: string | number): Observable<Array<Match>> {

    const options: RequestOptions = new RequestOptions({
      headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });

    const url: string = AppConfig.JOURNEY URL + '/' + idJourney +
AppConfig.MATCHES_URL;

    return this.http.get(url, options)
      .map((response: Response, index: number) =>
Match.toObjectArray(response.json()))
      .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

  /**
  * Get: return the list of matches of a journey with participants details
  * @return {Observable<Array<Match>>} returns the list of matches
  */
  public getMatchesJourneyDetails(idJourney: string | number, competition:
Competition): Observable<Array<Match>> {
    const ret: Array<Match> = new Array<Match>();
    let firstTime: boolean;

    return Observable.create(observer => {
      this.getMatchesJourney(idJourney).subscribe(matches => {
        matches.forEach(match => {
          if (competition.mode === 'Individual') {
          this.competitionService.getStudentsCompetition(competition.id)
          .subscribe( students => {
            firstTime = true;
            students.forEach(student => {
              if ( +student.id === match.playerOne ) {
                match.namePlayerOne = student.name + ' ' + student.surname;
              }
              if ( +student.id === match.playerTwo ) {
                match.namePlayerTwo = student.name + ' ' + student.surname;
              }
              if ( 0 === match.playerOne) {
                match.namePlayerOne = 'Ghost';
              }
              if ( 0 === match.playerTwo) {
                match.namePlayerTwo = 'Ghost';
              }
              if (match.namePlayerOne && match.namePlayerTwo && firstTime) {
                firstTime = false;
                ret.push(match);
              }
              if (ret.length === matches.length) {
```

```
                observer.next(ret);
                observer.complete();
              }
            });
          }, error => observer.error(error));
        } else {
          this.teamService.getTeamsCompetition(competition.id)
          .subscribe( teams => {
            firstTime = true;
            teams.forEach(team => {
              if ( +team.id === match.playerOne ) {
                match.namePlayerOne = team.name;
              }
              if ( +team.id === match.playerTwo ) {
                match.namePlayerTwo = team.name;
              }
              if ( 0 === match.playerOne) {
                match.namePlayerOne = 'Ghost';
              }
              if ( 0 === match.playerTwo) {
                match.namePlayerTwo = 'Ghost';
              }
              if (match.namePlayerOne && match.namePlayerTwo && firstTime) {
                firstTime = false;
                ret.push(match);
              }
              if (ret.length === matches.length) {
                observer.next(ret);
                observer.complete();
              }
            });
          }, error => observer.error(error));
        }
        });
    }, error => observer.error(error));
    });
  }
}
```

9.8.3. Matches.service.ts

```
export class MatchesService {

  constructor(public http: Http,
    public utilsService: UtilsService) { }

  /**
  * POST: add a new match
  * @return {Observable<Match>} returns the match
  */
  public postMatch (match: Match): Observable<Match> {

  const options: RequestOptions = new RequestOptions({
     headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
   });

   return this.http.post(AppConfig.MATCH_URL, match, options)
   .map((response: Response, index: number) => Match.toObject(response.json()))
   .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

  /**
  * PUT: add the winner in a match
  * @return {Observable<Match>} returns the match
  */
  public putWinner(winner: Match, matchId: string | number): Observable<Match> {

    const options: RequestOptions = new RequestOptions({
      headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });

    return this.http.put(AppConfig.MATCH URL + '/' + matchId, winner, options)
      .map((response: Response, index: number) => Match.toObject(response.json()))
```

```
      .catch((error: Response) => this.utilsService.handleAPIError(error));
  }
}
```

9.8.4. Team.service.ts

```
export class TeamService {

  constructor(
    public http: Http,
    public utilsService: UtilsService,
    public groupService: GroupService,
    public gradeService: GradeService,
    public matterService: MatterService,
    public competitionService: CompetitionService) { }

  /**
  * This method returns the list of competitions
  * of a team with the group (grade and matter)
  * @return {Observable<Array<Competition>>} returns the list of competitions
  */
  public getMyCompetitionsGroup(teamId: number): Observable<Array<Competition>> {

    const ret: Array<Competition> = new Array<Competition>();

    return Observable.create(observer => {
      this.getCompetitionsTeam(teamId).subscribe(competitions => {
        competitions.forEach(competition => {
          this.groupService.getGroup(competition.groupId).subscribe(
            group => {
              this.gradeService.getGrade(group.gradeId).subscribe(
                grade => {
                 competition.grade = grade;
                   this.matterService.getMatter(group.matterId).subscribe(
                     matter => {
                      competition.matter = matter;
                      ret.push(competition);
                      if (ret.length === competitions.length) {
                      observer.next(ret);
                      observer.complete();
                      }
                    }, error => observer.error(error));
                }, error => observer.error(error));
            }, error => observer.error(error));
        });
      }, error => observer.error(error));
    });
  }

  /**
  * This method returns the list of competitions of a team
  * @return {Observable<Array<Competition>>} returns the list of competitions
  */
  public getCompetitionsTeam(teamId: number): Observable<Array<Competition>> {
    const options: RequestOptions = new RequestOptions({
      headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });
  return this.http.get(AppConfig.TEAM_URL + '/' + teamId + AppConfig.COMPETITIONS_URL,
options)
  .map((response: Response, index: number) =>
Competition.toObjectArray(response.json()))
  .catch((error: Response) => this.utilsService.handleAPIError(error));
 }

   /**
  * POST: add a new team
  * @return {Observable<Team>} returns the team
  */
  public postTeam (team: Team): Observable<Team> {
    const options: RequestOptions = new RequestOptions({
     headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
```

```
    });
    return this.http.post(AppConfig.TEAM_URL, team, options)
    .map((response: Response, index: number) => Team.toObject(response.json()))
    .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

  /**
  * This method returns the list of teams of a competition
  * @return {Observable<Array<Team>>} returns the list of competitions
  */
  public getTeamsCompetition(competitionId: number | string): Observable<Array<Team>>
{
    const options: RequestOptions = new RequestOptions({
      headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
     });
    return this.http.get(AppConfig.COMPETITION_URL + '/' + competitionId +
AppConfig.TEAMS URL, options)
      .map((response: Response, index: number) => Team.toObjectArray(response.json()))
      .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

  /**
  * This method returns the list of students of a team
  * @return {Observable<Array<Student>>} returns the list of students
  */
  public getStudentsTeam(teamId: number): Observable<Array<Student>> {
    const options: RequestOptions = new RequestOptions({
     headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });
    return this.http.get(AppConfig.TEAM_URL + '/' + teamId + AppConfig.STUDENTS_URL,
options)
      .map((response: Response, index: number) =>
Student.toObjectArray(response.json()))
      .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

  /**
  * This method returns the list of teams of a student
  * @return {Observable<Array<Team>>} returns the list of teams
  */
  public getTeamsStudent(studentId: number): Observable<Array<Team>> {
    const options: RequestOptions = new RequestOptions({
      headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });
    return this.http.get(AppConfig.STUDENT_URL + '/' + studentId +
AppConfig.TEAMS_URL, options)
      .map((response: Response, index: number) => Team.toObjectArray(response.json()))
      .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

  /**
  * PUT: made the relation between a student and a team
  * @return {Observable<Response>}
  */
  public relTeamStudent (teamId: string | number, studentId: string | number):
Observable<Response> {
    const options: RequestOptions = new RequestOptions({
     headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });
    return this.http.put(AppConfig.TEAM_URL + '/' + teamId + AppConfig.STUDENTS_URL
      + AppConfig.REL_URL + '/' + studentId, Response , options)
      .map((response: Response) => response.json())
      .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

  /**
  * PUT: made the relation between a competition and a team
  * @return {Observable<Response>}
  */
  public relCompetitionTeam (competitionId: string | number, teamId: string | number):
Observable<Response> {
    const options: RequestOptions = new RequestOptions({
```

```
      headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });
    return this.http.put(AppConfig.COMPETITION_URL + '/' + competitionId +
AppConfig.TEAMS_URL
      + AppConfig.REL URL + '/' + teamId, Response , options)
      .map((response: Response) => response.json())
      .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

}
```

9.8.5. Group.service.ts

Solo han sido incluidos los métodos nuevos que han debido de añadirse a esta clase para realizar este proyecto, realmente incluye más métodos que han sido omitidos.

```
export class GroupService {
/**
   * Returns the information of the group by a group id
   * @return {Group} returns the group
   */
  public getGroup(id: number): Observable<Group> {

    const options: RequestOptions = new RequestOptions({
      headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
    });

    return this.http.get(AppConfig.GROUP_URL + '/' + id, options)
      .map((response: Response, index: number) => Group.toObject(response.json()))
      .catch((error: Response) => this.utilsService.handleAPIError(error));
  }

  /**
  * GET: Returns the list of teams of a group
  * @return {Observable<Array<Team>>} returns the list of teams
  */
  public getGroupTeams(groupId: string | number): Observable<Array<Team>> {

   const options: RequestOptions = new RequestOptions({
    headers: this.utilsService.setAuthorizationHeader(new Headers(),
this.utilsService.currentUser.id)
   });
   const url: string = AppConfig.GROUP_URL + '/' + groupId + AppConfig.TEAMS_URL;

   return this.http.get(url, options)
    .map((response: Response, index: number) => Team.toObjectArray(response.json()))
    .catch((error: Response) => this.utilsService.handleAPIError(error));
  }
}
```

9.9. Páginas de la aplicación móvil

Todos los siguientes apartados contendrán primero su archivo TypeScript, seguido de su plantilla HTML.

9.9.1. Competiciones

```
export class CompetitionsPage {

  public competitions: Array<Competition>;
```

```
  public myRole: Role;
  public role = Role;

  constructor(
    public ionicService: IonicService,
    public utilsService: UtilsService,
    public groupService: GroupService,
    public competitionService: CompetitionService,
    public translateService: TranslateService,
    public navCtrl: NavController,
    public navParams: NavParams) {
      this.competitions = [];
    }

  ionViewDidLoad() {
    this.ionicService.showLoading(this.translateService.instant('APP.WAIT'));
    this.myRole = this.utilsService.role;
    this.getMyCompetitionsInfo();
  }

  /**
   * This method returns the list of competitions from the
   * backend. This call is called on the ionViewDidLoad or the
   * refresh event
   * @param {Refresher} Refresher element
   */
  private getMyCompetitionsInfo(refresher?: Refresher): void {

    if (this.myRole === Role.SCHOOLADMIN) {
      this.ionicService.removeLoading();
    } else if (this.myRole === Role.TEACHER) {
        this.competitions = [];
        this.groupService.getMyGroups().finally(() => {
          refresher ? refresher.complete() : null;
          this.ionicService.removeLoading();
        }).subscribe(
          (( groups: Array<Group>) =>
           groups.map( group => {
            this.competitionService.getMyCompetitionsByGroup(group).subscribe(
             ((competitions: Array<Competition>) => {
              competitions.map( competition => {
                this.competitions.push(competition);
              })
            }),
            (error => {
              this.ionicService.removeLoading();
              this.ionicService.showAlert(this.translateService.instant('APP.ERROR'),
error);
            }));
          })),
          (error => {
            this.ionicService.removeLoading();
            this.ionicService.showAlert(this.translateService.instant('APP.ERROR'),
error);
          }));

    } else if (this.myRole === Role.STUDENT) {
      this.competitions = [];
      this.competitionService.getMyCompetitions().finally(() => {
        refresher ? refresher.complete() : null;
        this.ionicService.removeLoading();
      }).subscribe(
        ((competitions: Array<Competition>) => {
          competitions.map( competition => {
            this.competitions.push(competition);
          })
        }),
        (error => {
          this.ionicService.removeLoading();
          this.ionicService.showAlert(this.translateService.instant('APP.ERROR'),
error);
        }));

this.competitionService.getTeamsStudent(+this.utilsService.currentUser.userId).subscri
be(
        ((teams: Array<Team>) =>
        teams.map( team => {
```

```
            this.competitionService.getMyCompetitionsGroup(+team.id).subscribe(
              ((competitions: Array<Competition>) =>
              competitions.map( competition => {
                  this.competitions.push(competition);
              })),
              (error => {
                this.ionicService.removeLoading();

this.ionicService.showAlert(this.translateService.instant('APP.ERROR'), error);
              }));
        })),
        (error => {
          this.ionicService.removeLoading();
          this.ionicService.showAlert(this.translateService.instant('APP.ERROR'),
error);
        }));
        this.ionicService.removeLoading();
    }
  }

    /**
   * Method called from the competitions page to open
   * one competition of the list of competitions
   */
  private goToCompetition(competition) {
    competition.type === 'Liga' ?
    this.navCtrl.push(LeaguePage, {competition: competition}) :
    this.navCtrl.push(TennisPage, {competition: competition});
  }
}
```

9.9.2. Liga

```
export class LeaguePage {

  public competition: Competition;
  public showInfo: boolean;

  constructor(public navCtrl: NavController,
    public navParams: NavParams) {
    this.competition = this.navParams.get('competition');
    this.showInfo = false;
  }

  ionViewDidLoad() {
  }
  /**
   * Method called from the league page
   * to open the classification page
   */
  private gotoClassification() {
    this.navCtrl.push(ClassificationPage, {competition: this.competition})
  }
  /**
   * Method called from the league page
   * to open the calendar page
   */
  private gotoCalendar() {
    this.navCtrl.push(JourneysLeaguePage, {competition: this.competition})
  }
  /**
   * Method called from the league page
   * to open the teams page
   */
  private goToTeams() {
    this.navCtrl.push(TeamsPage, {competitionId: this.competition.id})
  }
  /**
   * Method called from the league page to show
   * or not show the information about the competition
   */
  private showInformation() {
    this.showInfo === false ? this.showInfo = true : this.showInfo = false;
```

```
  }

}
```

```
<ion-header>
  <ion-navbar color="primary">
    <button ion-button menuToggle>
      <ion-icon name="menu"></ion-icon>
    </button>
    <ion-title>{{ 'COMPETITIONS.LEAGUE' | translate }}</ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <div class="title">
    <h1 align="center">{{competition.name}}</h1>
  </div>
  <button class="btn" ion-button icon-left full (click)="gotoClassification()">
   <ion-icon class="btn-icon" name="trophy"></ion-icon>
   {{ 'CLASSIFICATION.TITLE' | translate }}
  </button>
  <button class="btn" ion-button icon-left full (click)="gotoCalendar()">
    <ion-icon class="btn-icon" name="calendar"></ion-icon>
    {{ 'COMPETITIONS.CALENDAR' | translate }}
  </button>
  <button *ngIf="competition.mode === 'Equipos'" (click)="goToTeams()" class="btn" ion-
button icon-left full>
    <ion-icon class="btn-icon" name="people"></ion-icon>
    {{ 'COMPETITIONS.TEAMS' | translate }}
  </button>
  <button     *ngIf="!showInfo"     class="btn"     ion-button     icon-left     full
(click)="showInformation()">
    <ion-icon class="btn-icon" name="information-circle"></ion-icon>
     {{ 'COMPETITIONS.SHOW_INFO' | translate }}
  </button>
  <button     *ngIf="showInfo"     class="btn"     ion-button     icon-left     full
(click)="showInformation()">
    <ion-icon class="btn-icon" name="information-circle"></ion-icon>
    {{ 'COMPETITIONS.HIDE INFO' | translate }}
  </button>
  <ion-card *ngIf="showInfo">
    <ion-card-header>
        {{ 'COMPETITIONS.INFO COMP' | translate }}:
    </ion-card-header>
    <ion-card-content>
      {{competition.information}}
    </ion-card-content>
  </ion-card>
</ion-content>
```

9.9.3. Clasificación de la liga

```
export class ClassificationPage {

  public competition: Competition;
  public journeys: Array<Journey>;
  public matchesJourneys: Array<Array<Match>>;
  public participants: Array<Participant>;

  public scores: Array<Score>;
  public odd: boolean;

  constructor(
    public navCtrl: NavController,
    public navParams: NavParams,
    public ionicService: IonicService,
    public translateService: TranslateService,
    public competitionService: CompetitionService,
    public journeyService: JourneyService) {
    this.competition = this.navParams.get('competition');
  }
```

```
  ionViewDidLoad() {
    this.ionicService.showLoading(this.translateService.instant('APP.WAIT'));
    this.getJourneys();
  }
  /**
   * This method returns the journeys of the current competition
   * and calls the getMatches method
   */
  private getJourneys(refresher?: Refresher): void {
    this.journeyService.getJourneysCompetition(this.competition.id).finally(() => {
      refresher ? refresher.complete() : null;
    }).subscribe(
      ((journeys: Array<Journey>) => {
        this.journeys = journeys;
        this.getMatches();
      }),
      (error => {
        this.ionicService.removeLoading();
        this.ionicService.showAlert(this.translateService.instant('APP.ERROR'),
error);
      }));
  }
  /**
   * This method returns the matches of each journey
   * and calls the getParticipants method
   */
  private getMatches(): void {
   let countJourneys = 0;
    this.matchesJourneys = [];
    for (let _j = 0; _j < this.journeys.length; _j++) {
      this.matchesJourneys[_j] = [];
      this.journeyService.getMatchesJourneyDetails(this.journeys[_j].id,
this.competition).subscribe(
      ((matches: Array<Match>) => {
        countJourneys ++;
        for (let _m = 0; _m < matches.length; _m++) {
          this.matchesJourneys[_j][_m] = new Match();
          this.matchesJourneys[_j][_m] = matches[_m];
        }
         if ( countJourneys === this.journeys.length ) {
           this.getParticipants();
          }
      }),
      (error => {
        this.ionicService.removeLoading();
        this.ionicService.showAlert(this.translateService.instant('APP.ERROR'),
error);
      }));
    }
  }
  /**
   * This method returns the participants of the current competition
   * and calls the getScores method
   */
  private getParticipants(): void {
    this.participants = [];
    if (this.competition.mode === 'Individual') {
      this.competitionService.getStudentsCompetition(this.competition.id)
      .subscribe(( (students: Array<Student>) => {
        students.length % 2 === 0 ? this.odd = false : this.odd = true;
        for (let _s = 0; _s < students.length; _s++) {
          this.participants[ s] = {
            id: +students[ s].id,
            name: students[_s].name.concat(' ', students[_s].surname)
          };
        }
        this.getScores();
      }),
      (error => {
        this.ionicService.removeLoading();
        this.ionicService.showAlert(this.translateService.instant('APP.ERROR'),
error);
      }));
      } else {
      this.competitionService.getTeamsCompetition(this.competition.id)
      .subscribe(( (teams: Array<Team>) => {
        teams.length % 2 === 0 ? this.odd = false : this.odd = true;
```

```
        for (let  t = 0;  t < teams.length;  t++) {
          this.participants[_t] = {
            id: +teams[_t].id,
            name: teams[_t].name
          };
        }
        this.getScores();
      }),
      (error => {
        this.ionicService.removeLoading();
        this.ionicService.showAlert(this.translateService.instant('APP.ERROR'),
error);
      }));
      }
  }
  /**
   * This method computes the score of each participant
   * (position, points and played, won, draw and lost games)
   */
  private getScores(): void {
      this.scores = [];
      for (let  p = 0;  p < this.participants.length;  p++) {
        let score = { position: 0, name: this.participants[ p].name,
                      played: 0, won: 0, draw: 0, lost: 0, points: 0};
        for (let _j = 0; _j < this.journeys.length; _j++) {
          let found = false;
          for (let  m = 0;  m < this.matchesJourneys[ j].length && !found;  m++) {
            if ( +this.participants[ p].id === this.matchesJourneys[ j][ m].playerOne
||
            +this.participants[_p].id === this.matchesJourneys[_j][_m].playerTwo ) {
              if ( this.matchesJourneys[_j][_m].winner === +this.participants[_p].id )
{
                score.points = score.points + 3;
                score.won = score.won + 1;
                score.played = score.played + 1;
              } else if ( this.matchesJourneys[_j][_m].winner === 1 ) {
                score.points = score.points + 1;
                score.draw = score.draw + 1;
                score.played = score.played + 1;
              } else if ( this.matchesJourneys[ j][ m].winner === 2
              || this.matchesJourneys[ j][ m].winner === 0 ) {
              } else {
                score.lost = score.lost + 1;
                score.played = score.played + 1;
              }
              found = true;
            }
          }
        }
        this.scores.push(score);
      }
      this.scores.sort(function (a, b) {
        return (b.points - a.points);
      });
      for (let _s = 0; _s < this.scores.length; _s++) { this.scores[_s].position = _s
+ 1; }
      this.ionicService.removeLoading();
  }

}

export interface Score {
  position: number;
  name: string;
  played: number;
  won: number;
  draw: number;
  lost: number;
  points: number;
}

export interface Participant {
  id: number;
  name: string;
}
```

```
<ion-header>
  <ion-navbar color="primary">
    <button ion-button menuToggle>
      <ion-icon name="menu"></ion-icon>
    </button>
    <ion-title>{{ 'CLASSIFICATION.TITLE' | translate }}</ion-title>
  </ion-navbar>
</ion-header>


<ion-content padding>
 <ion-refresher (ionRefresh)="getJourneys($event)">
  <ion-refresher-content></ion-refresher-content>
 </ion-refresher>
  <table>
    <caption   *ngIf="competition">{{   'CLASSIFICATION.LEAGUE'   |   translate   }}
{{competition.name}}</caption>
    <thead>
      <tr class="tableHeader">
        <th class="position">{{ 'CLASSIFICATION.N_S' | translate }}</th>
        <th class="name">{{ 'CLASSIFICATION.PARTICIPANT' | translate }}</th>
        <th>{{ 'CLASSIFICATION.PLAYED S' | translate }}</th>
        <th>{{ 'CLASSIFICATION.WON S' | translate }}</th>
        <th>{{ 'CLASSIFICATION.DRAW S' | translate }}</th>
        <th>{{ 'CLASSIFICATION.LOST_S' | translate }}</th>
        <th>{{ 'CLASSIFICATION.POINTS_S' | translate }}</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let score of scores ; let i=index;">
        <td id="position_{{i + 1}}" class="position">{{score.position}}</td>
        <td class="name">{{score.name}}</td>
        <td>{{score.played}}</td>
        <td>{{score.won}}</td>
        <td>{{score.draw}}</td>
        <td>{{score.lost}}</td>
        <td class="points">{{score.points}}</td>
      </tr>
    </tbody>
  </table>
  <div class="legend">
  <ion-grid>
  <ion-row>
    <ion-col>{{ 'CLASSIFICATION.N_S' | translate }} : {{ 'CLASSIFICATION.N' | translate
}}</ion-col>
    <ion-col>{{ 'CLASSIFICATION.PLAYED S' | translate }} : {{ 'CLASSIFICATION.PLAYED'
| translate }}</ion-col>
    <ion-col>{{  'CLASSIFICATION.WON_S'  |  translate  }}  :  {{  'CLASSIFICATION.WON'  |
translate }}</ion-col>
  </ion-row>
  <ion-row>
    <ion-col>{{  'CLASSIFICATION.DRAW S' | translate }}  : {{ 'CLASSIFICATION.DRAW' |
translate }}</ion-col>
    <ion-col>{{  'CLASSIFICATION.LOST_S' | translate }} : {{  'CLASSIFICATION.LOST' |
translate }}</ion-col>
    <ion-col>{{ 'CLASSIFICATION.POINTS S' | translate }} : {{ 'CLASSIFICATION.POINTS'
| translate }}</ion-col>
  </ion-row>
</ion-grid>
  </div>
  <div *ngIf="odd" class="information">
    {{ 'CLASSIFICATION.INFORMATION' | translate }}
  </div>
</ion-content>
```

9.9.4. Calendario de la liga

```
export class JourneysLeaguePage {

  public competition: Competition;
  public journeys: Array<Journey>;
  public matchesJourneys: Array<Array<Match>>;
```

```
  public descanso: number;
  public dates: Array<String>;

  constructor(
    public navCtrl: NavController,
    public navParams: NavParams,
    public ionicService: IonicService,
    public utilsService: UtilsService,
    public translateService: TranslateService,
    public competitionService: CompetitionService,
    public journeyService: JourneyService,
    public datePipe: DatePipe) {
    this.competition = this.navParams.get('competition');
  }

  ionViewDidLoad() {
    this.ionicService.showLoading(this.translateService.instant('APP.WAIT'));
    this.getJourneys();
  }
  /**
   * This method returns the journeys of the current competition
   * and calls the getMatches method
   */
  private getJourneys(refresher?: Refresher): void {
    this.journeyService.getJourneysCompetition(this.competition.id).finally(() => {
      refresher ? refresher.complete() : null;
    }).subscribe(
      ((journeys: Array<Journey>) => {
        this.journeys = journeys;
        this.journeys.sort(function (a, b) {
          return (a.number - b.number);
        });
        this.getMatches();
      }),
      (error => {
        this.ionicService.removeLoading();
        this.ionicService.showAlert(this.translateService.instant('APP.ERROR'),
error);
      }));
  }
  /**
   * This method returns the matches of each journey
   * and calls the getDatesAndResults method
   */
  private getMatches(): void {
    let countJourneys = 0;
    this.matchesJourneys = [];
    for (let _j = 0; _j < this.journeys.length; _j++) {
      this.matchesJourneys[_j] = [];
      this.journeyService.getMatchesJourneyDetails(this.journeys[ j].id,
this.competition).subscribe(
      ((matches: Array<Match>) => {
        countJourneys = countJourneys + 1;
        for (let _m = 0; _m < matches.length; _m++) {
          if (matches[_m].namePlayerOne === 'Ghost' || matches[_m].namePlayerTwo ===
'Ghost') {
            this.descanso =  m;
          }
        }
        if (this.descanso !== undefined) {
          matches.splice(this.descanso,  1);  //  y  ocultando  el  enfrentamiento  del
descanso
          }
        this.matchesJourneys[_j] = matches;
         if ( countJourneys === this.journeys.length ) {
           this.getDatesAndResults();
          }
      }),
      (error => {
        this.ionicService.removeLoading();
        this.ionicService.showAlert(this.translateService.instant('APP.ERROR'),
error);
      }));
    }
  }
  /**
   * This method make the date and results of each journey
```

```
   * to show in the journeys-league page
   */
  private getDatesAndResults(): void {
    this.dates = [];
    for (let _j = 0; _j < this.journeys.length; _j++) {
      this.journeys[ j].date === null ?
        this.dates[ j] = this.translateService.instant('TOURNAMENTS.NOT ESTABLISHED')
:
        this.dates[_j] = this.datePipe.transform(this.journeys[_j].date, 'dd-MM-
yyyy');
      for (let _m = 0; _m < this.matchesJourneys[_j].length; _m++) {
        if (this.matchesJourneys[ j][ m].winner ===
this.matchesJourneys[ j][ m].playerOne ) {
          this.matchesJourneys[_j][_m].result =
this.matchesJourneys[_j][_m].namePlayerOne;
        } else if ( this.matchesJourneys[_j][_m].winner ===
this.matchesJourneys[_j][_m].playerTwo ) {
          this.matchesJourneys[ j][ m].result =
this.matchesJourneys[ j][ m].namePlayerTwo;
        } else if ( this.matchesJourneys[_j][_m].winner === 1 ) {
          this.matchesJourneys[_j][_m].result = 'Empate';
        } else if ( this.matchesJourneys[ j][ m].winner === 0 ) {
          this.matchesJourneys[ j][ m].result = '-';
        }
      }
    }
    this.ionicService.removeLoading();
  }
}
```

```
<ion-header>
  <ion-navbar color="primary">
    <button ion-button menuToggle>
      <ion-icon name="menu"></ion-icon>
    </button>
    <ion-title>{{ 'COMPETITIONS.CALENDAR' | translate }}</ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <ion-refresher (ionRefresh)="getJourneys($event)">
    <ion-refresher-content></ion-refresher-content>
  </ion-refresher>
  <div *ngIf="dates">
    <h4>{{  'COMPETITIONS.CALENDAR OF'  |  translate  }}  {{  'COMPETITIONS.LEAGUE'  |
translate }}: {{ competition.name }}</h4>
        <ion-card class="journey-card" *ngFor="let date of dates; let i=index;">
          <ion-card-content>
              <p><b>{{ 'COMPETITIONS.JOURNEY' | translate }} {{i + 1}}</b></p>
           <table>
              <caption>
                {{ 'COMPETITIONS.DATE' | translate }}: {{date}}
              </caption>
              <thead>
                <tr class="tableHeader">
                  <th   *ngIf="competition.mode   ==   'Individual'"   class="player">{{
'COMPETITIONS.PLAYER' | translate }} 1</th>
                  <th   *ngIf="competition.mode   ==   'Individual'"   class="player">{{
'COMPETITIONS.PLAYER' | translate }} 2</th>
                   <th    *ngIf="competition.mode    ==    'Equipos'"    class="player">{{
'COMPETITIONS.TEAM' | translate }} 1</th>
                  <th    *ngIf="competition.mode    ==    'Equipos'"    class="player">{{
'COMPETITIONS.TEAM' | translate }} 2</th>
                  <th class="result">{{ 'COMPETITIONS.WINNER' | translate }}</th>
                </tr>
              </thead>
              <tbody>
                <tr *ngFor="let score of matchesJourneys[i]">
                  <td class="player">{{score.namePlayerOne}}</td>
                  <td class="player">{{score.namePlayerTwo}}</td>
                  <td class="result">{{score.result}}</td>
                </tr>
              </tbody>
            </table>
          </ion-card-content>
        </ion-card>
```

```
    </div>
</ion-content>
```

9.9.5. Torneo de tenis

```
export class TennisPage {

  public competition: Competition;
  public showInfo: boolean;

  constructor(public navCtrl: NavController, public navParams: NavParams) {
    this.competition = this.navParams.get('competition');
    this.showInfo = false;
  }

  ionViewDidLoad() {
  }
  /**
   * Method called from the tennis page
   * to open the tournaments page
   */
  private goToTournaments() {
    this.navCtrl.push(TournamentsPage, {competition: this.competition})
  }
  /**
   * Method called from the tennis page
   * to open the calendar page
   */
  private goToCalendar() {
    this.navCtrl.push(JourneysTennisPage, {competition: this.competition})
  }
  /**
   * Method called from the tennis page
   * to open the teams page
   */
  private goToTeams() {
    this.navCtrl.push(TeamsPage, {competitionId: this.competition.id})
  }
  /**
   * Method called from the tennis page to show
   * or not show the information about the competition
   */
  private showInformation() {
    this.showInfo === false ? this.showInfo = true : this.showInfo = false;
  }
}
```

```
<ion-header>
    <ion-navbar color="primary">
        <button ion-button menuToggle>
          <ion-icon name="menu"></ion-icon>
        </button>
        <ion-title>{{ 'TENNIS.TITLE' | translate }}</ion-title>
      </ion-navbar>
</ion-header>

<ion-content padding>
  <div class="title">
    <h1 align="center">{{competition.name}}</h1>
  </div>
  <button class="btn" ion-button (click)="goToTournaments()" icon-left full>
   <ion-icon class="btn-icon" name="trophy"></ion-icon>
   {{ 'TENNIS.TOURNAMENT_TRACKING' | translate }}
  </button>
  <button class="btn" ion-button (click)="goToCalendar()" icon-left full>
    <ion-icon class="btn-icon" name="calendar"></ion-icon>
    {{ 'COMPETITIONS.CALENDAR' | translate }}
  </button>
  <button *ngIf="competition.mode === 'Equipos'" (click)="goToTeams()" class="btn" ion-
button icon-left full>
    <ion-icon class="btn-icon" name="people"></ion-icon>
    {{ 'COMPETITIONS.TEAMS' | translate }}
```

```
  </button>
  <button     *ngIf="!showInfo"     class="btn"     ion-button     icon-left     full
(click)="showInformation()">
    <ion-icon class="btn-icon" name="information-circle"></ion-icon>
    {{ 'COMPETITIONS.SHOW_INFO' | translate }}
  </button>
  <button     *ngIf="showInfo"     class="btn"     ion-button     icon-left     full
(click)="showInformation()">
    <ion-icon class="btn-icon" name="information-circle"></ion-icon>
    <b>{{ 'COMPETITIONS.HIDE_INFO' | translate }}</b>
  </button>
  <ion-card *ngIf="showInfo">
    <ion-card-header>
      <b>{{ 'COMPETITIONS.INFO_COMP' | translate }}:</b>
    </ion-card-header>
    <ion-card-content>
      {{competition.information}}
    </ion-card-content>
  </ion-card>

</ion-content>
```

9.9.6. Seguimiento del torneo

```
export class TournamentsPage {

  public final = false;
  public finished = false;
  public tournamentCompleted = false;
  public winner: string;

  public competition: Competition;
  public journeys: Array<Journey>;
  public matchesJourneys: Array<Array<Match>>;

  public lastJourney: number;
  public participants: any[];
  public participantsPrimary: String[];
  public participantsSecondary: String[];
  public participantsEliminated: String[];
  ghostIndex: number;

  constructor(public navCtrl: NavController,
    public navParams: NavParams,
    public ionicService: IonicService,
    public translateService: TranslateService,
    public competitionService: CompetitionService,
    public journeyService: JourneyService) {
      this.competition = this.navParams.get('competition');
    }

  ionViewDidLoad() {
    this.ionicService.showLoading(this.translateService.instant('APP.WAIT'));
    this.getJourneys();
  }
  /**
   * This method returns the journeys of the current competition
   * and calls the getMatches method
   */
  private getJourneys(): void {
    this.journeyService.getJourneysCompetition(this.competition.id).subscribe(
      ((journeys: Array<Journey>) => {
        this.journeys = journeys;
        this.journeys.sort(function (a, b) { return (a.number - b.number); });
        this.getMatches();
      }), (error => {
        this.ionicService.removeLoading();
        this.ionicService.showAlert(this.translateService.instant('APP.ERROR'),
error);
      }));
  }
  /**
   * This method returns the matches of each journey
   * and calls the getParticipants method
   */
```

```
  private getMatches(): void {
    this.matchesJourneys = [];
    let journeysCompleted = 0;
    for (let _n = 0; _n < this.journeys.length; _n++) {
      this.journeyService.getMatchesJourneyDetails(this.journeys[_n].id,
this.competition).subscribe(
      ((matches: Array<Match>) => {
        this.matchesJourneys[_n] = [];
        for (let _m = 0; _m < matches.length; _m++) {
          this.matchesJourneys[_n][_m] = new Match();
          this.matchesJourneys[_n][_m] = matches[_m];
        }
        journeysCompleted++;
        if (this.matchesJourneys[_n][0].winner === 0) { this.lastJourney = _n; }
        if ( journeysCompleted === this.lastJourney + 1 || this.matchesJourneys.length
=== this.journeys.length) {
          if    (   this.lastJourney   ===   undefined   )   {   this.lastJourney   =
this.journeys.length - 1; }
          this.getParticipants();
        }
      }), (error => {
        this.ionicService.removeLoading();
        this.ionicService.showAlert(this.translateService.instant('APP.ERROR'),
error);
      }));
    }
  }
  /**
   * This method returns the participants of the current competition
   * and calls the getTournamentStatus method
   */
  private getParticipants(): void {
    this.participants = [];
    if (this.competition.mode === 'Individual') {
      this.competitionService.getStudentsCompetition(this.competition.id)
      .subscribe(( (students: Array<Student>) => {
        for (let _s = 0; _s < students.length; _s++) {
          this.participants[_s] = {
            id: +students[ s].id,
            name: students[ s].name.concat(' ', students[ s].surname)
          };
        }
        this.getTournamentStatus();
      }), (error => {
        this.ionicService.removeLoading();
        this.ionicService.showAlert(this.translateService.instant('APP.ERROR'),
error);
      }));
      } else {
      this.competitionService.getTeamsCompetition(this.competition.id)
      .subscribe(( (teams: Array<Team>) => {
        for (let  t = 0;  t < teams.length;  t++) {
          this.participants[_t] = {
            id: +teams[_t].id,
            name: teams[_t].name
          };
        }
        this.getTournamentStatus();
      }), (error => {
        this.ionicService.removeLoading();
        this.ionicService.showAlert(this.translateService.instant('APP.ERROR'),
error);
      }));
      }
  }
  /**
   * This method divides the participants between the main tournament,
   *  the secondary tournament and the eliminated ones
   */
  private getTournamentStatus(): void {

   this.participantsPrimary = [];
   this.participantsSecondary = [];

    for (let _m = 0; _m < this.matchesJourneys[this.lastJourney].length; _m++) {
     if ( this.lastJourney === 0 ) {
```

```
this.participantsPrimary.push(this.matchesJourneys[this.lastJourney][_m].namePlayerOne
);

this.participantsPrimary.push(this.matchesJourneys[this.lastJourney][_m].namePlayerTwo
);
     } else if ( (this.lastJourney + 1) % 2 === 0 && this.lastJourney + 1 !==
this.journeys.length ) {
       if ( _m < this.matchesJourneys[this.lastJourney].length / 2 ) {

this.participantsPrimary.push(this.matchesJourneys[this.lastJourney][_m].namePlayerOne
);

this.participantsPrimary.push(this.matchesJourneys[this.lastJourney][_m].namePlayerTwo
);
       } else {

this.participantsSecondary.push(this.matchesJourneys[this.lastJourney][ m].namePlayerO
ne);

this.participantsSecondary.push(this.matchesJourneys[this.lastJourney][_m].namePlayerT
wo);
       }
     } else if ( (this.lastJourney + 1) % 2 !== 0 ) {
      this.matchesJourneys[this.lastJourney        -        1][_m].winner        ===
this.matchesJourneys[this.lastJourney - 1][_m].playerOne ?
      this.participantsPrimary.push(this.matchesJourneys[this.lastJourney           -
1][ m].namePlayerOne) :
      this.participantsPrimary.push(this.matchesJourneys[this.lastJourney           -
1][_m].namePlayerTwo);

this.participantsSecondary.push(this.matchesJourneys[this.lastJourney][_m].namePlayerO
ne);

this.participantsSecondary.push(this.matchesJourneys[this.lastJourney][ m].namePlayerT
wo);
     } else if ( this.lastJourney + 1 === this.journeys.length ) {
      this.final = true;
      if ( this.matchesJourneys[this.lastJourney][0].winner !== 0) {
        this.tournamentCompleted = true;
        this.matchesJourneys[this.lastJourney][0].winner                            ===
this.matchesJourneys[this.lastJourney][0].playerOne ?
        this.winner = this.matchesJourneys[this.lastJourney][0].namePlayerOne :
        this.winner = this.matchesJourneys[this.lastJourney][0].namePlayerTwo;
      }

this.participantsPrimary.push(this.matchesJourneys[this.lastJourney][ m].namePlayerOne
);

this.participantsPrimary.push(this.matchesJourneys[this.lastJourney][ m].namePlayerTwo
);
     }
    }

    // Deleting ghosts to show
    this.ghostIndex = 0;
    while ( this.ghostIndex < this.participantsPrimary.length) {
      if (this.participantsPrimary[this.ghostIndex] === 'Ghost') {
        this.participantsPrimary.splice(this.ghostIndex, 1);
        this.ghostIndex = 0;
      } else { this.ghostIndex++; }
    }
    this.ghostIndex = 0;
    while ( this.ghostIndex < this.participantsSecondary.length) {
      if (this.participantsSecondary[this.ghostIndex] === 'Ghost') {
        this.participantsSecondary.splice(this.ghostIndex, 1);
        this.ghostIndex = 0;
      } else { this.ghostIndex++; }
    }

    // Adding eliminated participants
    this.participantsEliminated = [];
    for (let  d = 0;  d < this.participants.length;  d++) {
      let count = 0;
      for (let _p = 0; _p < this.participantsPrimary.length; _p++) {
        this.participants[_d].name === this.participantsPrimary[_p] ? count = 1 : null;
      }
```

```
      count === 0 ?  this.participantsEliminated.push(this.participants[ d].name) :
null;
    }

    let _q = 0;
    while ( q < this.participantsEliminated.length) {
      let count = 0;
      for (let _p = 0; _p < this.participantsSecondary.length; _p++) {
        this.participantsEliminated[_q] === this.participantsSecondary[_p] ? count = 1
: null;
      }
      if (count === 1) {
       this.participantsEliminated.splice( q, 1);
       _q = 0;
      } else { _q++; }
    }
    this.ionicService.removeLoading();
    this.finished = true;
  }
}
```

```
<ion-header>
  <ion-navbar color="primary">
    <button ion-button menuToggle>
      <ion-icon name="menu"></ion-icon>
    </button>
    <ion-title>{{ 'TENNIS.TOURNAMENT TRACKING' | translate }}</ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <div *ngIf="finished">
    <h2  class="title2">{{  'TENNIS.TOURNAMENT TRACKING'  |  translate  }}:  <b>{{
competition.name }}</b></h2>
    <ion-card *ngIf="winner" class="winner-card">
      <h3><b>{{ 'TOURNAMENTS.FINISHED' | translate }}.</b></h3>
      <h1><p   class="winner"><b>{{   'COMPETITIONS.WINNER'   |   translate   }}:
{{winner}}</b></p></h1>
    </ion-card>
    <h4  *ngIf="!winner"><b>{{  'TOURNAMENTS.PARTICIPANTS'  |  translate  }}  ({{
'COMPETITIONS.JOURNEY_NUMBER' | translate }} {{lastJourney + 1}}):</b></h4>
    <div *ngIf="!final">
    <ion-card class="tournaments-card">
      <ion-card-content>
        <ion-list>
          <h2 class="subtitle"> {{ 'TOURNAMENTS.PRINCIPAL' | translate }}</h2>
          <ion-item *ngFor="let primary of participantsPrimary">
            <ion-icon  *ngIf="competition.mode  ===  'Individual'"  name="person"></ion-
icon>
            <ion-icon *ngIf="competition.mode === 'Equipos'" name="people"></ion-icon>
          <span>{{primary}}</span>
          </ion-item>
          <ion-item-divider></ion-item-divider>
          <h2 class="subtitle">{{ 'TOURNAMENTS.SECONDARY' | translate }}</h2>
          <ion-item *ngFor="let secondary of participantsSecondary">
            <ion-icon  *ngIf="competition.mode  ===  'Individual'"  name="person"></ion-
icon>
            <ion-icon *ngIf="competition.mode === 'Equipos'" name="people"></ion-icon>
            <span>{{secondary}}</span>
          </ion-item>
          <h2  class="noSecondPlayers"  *ngIf="participantsSecondary.length  ===  0">{{
'TOURNAMENTS.NO_PLAYER_SECONDARY' | translate }}.</h2>
        </ion-list>
      </ion-card-content>
    </ion-card>
  </div>
  <div *ngIf="final">
   <h4 class="final"><b>{{ 'TOURNAMENTS.FINAL' | translate }}:</b></h4>
     <ion-card class="eliminated-card">
       <ion-card-content>
         <ion-list>
           <h2 class="subtitle">{{ 'TOURNAMENTS.FINALISTS' | translate }}</h2>
           <ion-item *ngFor="let final of participantsPrimary">
            <ion-icon  *ngIf="competition.mode  ===  'Individual'"  name="person"></ion-
icon>
            <ion-icon *ngIf="competition.mode === 'Equipos'" name="people"></ion-icon>
```

```
            <span>{{final}}</span>
          </ion-item>
        </ion-list>
      </ion-card-content>
    </ion-card>
  </div>
  <div>
    <h4><b>{{ 'TOURNAMENTS.PLAYERS_ELIMINATED' | translate }}:</b></h4>
    <ion-card class="eliminated-card">
      <ion-card-content>
        <ion-list>
          <h2 class="subtitle">{{ 'TOURNAMENTS.ELIMINATED' | translate }}</h2>
          <ion-item *ngFor="let eliminated of participantsEliminated">
            <ion-icon *ngIf="competition.mode === 'Individual'" name="person"></ion-
icon>
            <ion-icon  *ngIf="competition.mode  ===  'Equipos'"  name="people"></ion-
icon>
            <span>{{eliminated}}</span>
          </ion-item>
          <ion-item       *ngIf="participantsEliminated.length       ===       0">{{
'TOURNAMENTS.NO_PLAYER_ELIMINATED' | translate }}.</ion-item>
        </ion-list>
      </ion-card-content>
    </ion-card>
    </div>
  </div>
</ion-content>
```

9.9.7. Calendario del torneo de tenis

```
export class JourneysTennisPage {

  show : boolean;
  results : boolean;
  competition: Competition;
  journeys = new Array<Journey>();
  numJourneys: number;
  dates: String[];
  datesNoMatches: any[];

  matchesJourneys: Match[][];
  matchesPrincipal: Match[][];
  matchesSecondary: Match[][];
  lastJourney: number;
  tournaments: String[][];

  constructor(
    public navCtrl: NavController,
    public navParams: NavParams,
    public ionicService: IonicService,
    public utilsService: UtilsService,
    public translateService: TranslateService,
    public competitionService: CompetitionService,
    public journeyService: JourneyService,
    public datePipe: DatePipe) {
      this.competition = this.navParams.get('competition');
      this.show = false;
      this.results = false;
    }

  ionViewDidLoad() {
    this.ionicService.showLoading(this.translateService.instant('APP.WAIT'));
    this.getJourneys();
  }
  /**
   * This method returns the journeys of the current competition
   * and calls the getMatches method
   */
  private getJourneys(): void {
    this.journeyService.getJourneysCompetition(this.competition.id).subscribe(
      ((journeys: Array<Journey>) => {
        this.journeys = journeys;
        this.journeys.sort(function (a, b) {
```

```
          return (a.number - b.number);
        });
        this.numJourneys = this.journeys.length;
        this.getMatches();
      }),
      (error => {
        this.ionicService.removeLoading();
        this.ionicService.showAlert(this.translateService.instant('APP.ERROR'),
error);
      }));
  }
  /**
   * This method returns the matches of each journey
   * and calls the getDatesAndResults method
   */
  private getMatches(): void {
    let countJourneys = 0;
    this.matchesJourneys = [];
    for (let  j = 0;  j < this.journeys.length;  j++) {
      this.journeyService.getMatchesJourneyDetails(this.journeys[_j].id,
this.competition).subscribe(
      ((matches: Array<Match>) => {
        this.matchesJourneys[ j] = [];
        for (let  m = 0;  m < matches.length;  m++) {
          this.matchesJourneys[_j][_m] = new Match();
          this.matchesJourneys[_j][_m] = matches[_m];
        }
        if (this.matchesJourneys[ j][0].winner === 0) { this.lastJourney =  j; }
        countJourneys++;
        if  (  countJourneys  ===  this.lastJourney  +  1  ||  countJourneys  ===
this.numJourneys) {
          if ( this.lastJourney === undefined ) { this.lastJourney = this.numJourneys
- 1; }
          this.getDatesAndResults();
        }
      }),
      (error => {
        this.ionicService.removeLoading();
        this.ionicService.showAlert(this.translateService.instant('APP.ERROR'),
error);
      }));
    }
  }
  /**
   * This method make the date and results of each journey
   * and the rest of content to show in the journeys-tennis page
   */
  private getDatesAndResults(): void {
    this.dates = [];
    this.datesNoMatches = [];
    this.tournaments = [];
    // DATES AND TOURNAMENTS
    for (let _j = 0; _j < this.journeys.length; _j++) {
      if ( _j <= this.lastJourney) {
      this.journeys[_j].date === null ?
        this.dates[ j]  =  this.translateService.instant('TOURNAMENTS.NOT ESTABLISHED')
:
        this.dates[ j]   =   this.datePipe.transform(this.journeys[ j].date,   'dd-MM-
yyyy');
      } else {
        this.journeys[_j].date === null ?
        this.datesNoMatches.push({date:
this.translateService.instant('TOURNAMENTS.NOT ESTABLISHED'), number:  j + 1}) :
        this.datesNoMatches.push({date:
this.datePipe.transform(this.journeys[_j].date, 'dd-MM-yyyy'), number: _j + 1});
        if ((_j + 1) % 2 === 0 && _j !== this.journeys.length - 1) { // si es par y no
es el final participa en ambos
          this.tournaments.push([
            this.translateService.instant('TOURNAMENTS.PRINCIPAL')    +    ':    '    +
this.translateService.instant('TOURNAMENTS.PARTICIPATES'),
            this.translateService.instant('TOURNAMENTS.SECONDARY')    +    ':    '    +
this.translateService.instant('TOURNAMENTS.PARTICIPATES')]);
        } else if (( j + 1) % 2 !== 0) {
          this.tournaments.push([
            this.translateService.instant('TOURNAMENTS.PRINCIPAL')    +    ':    '    +
this.translateService.instant('TOURNAMENTS.DONT_PARTICIPATES'),
```

```
            this.translateService.instant('TOURNAMENTS.SECONDARY')    +    ':    '    +
this.translateService.instant('TOURNAMENTS.PARTICIPATES')]);
        } else if ( _j === this.journeys.length - 1) {
          this.tournaments.push([this.translateService.instant('TOURNAMENTS.FINAL')]);
        }
      }
    }
    // TO INTRODUCE THE RESULT AND TO SEPARATE IN PRINCIPAL AND SECONDARY MATCHES
    this.matchesPrincipal = [];
    this.matchesSecondary = [];
    for (let _j = 0; _j < this.matchesJourneys.length; _j++) {
      this.matchesPrincipal[ j] = [];
      this.matchesSecondary[ j] = [];
      for (let _m = 0; _m < this.matchesJourneys[_j].length; _m++) {
        if ( this.matchesJourneys[_j][_m].winner === 0 ) {
          this.matchesJourneys[_j][_m].result = '-';
        } else {
          this.matchesJourneys[ j][ m].winner                                          ===
this.matchesJourneys[ j][ m].playerOne ?
          this.matchesJourneys[_j][_m].result                                           =
this.matchesJourneys[_j][_m].namePlayerOne :
          this.matchesJourneys[ j][ m].result                                           =
this.matchesJourneys[ j][ m].namePlayerTwo;
        }
        if ((_j + 1) % 2 === 0) { // si es par participa en ambos
          _m < this.matchesJourneys[_j].length / 2 ?
          this.matchesPrincipal[ j].push(this.matchesJourneys[ j][ m]) :
          this.matchesSecondary[ j].push(this.matchesJourneys[ j][ m]);
        } else if (( j + 1) % 2 !== 0 &&  j !== 0) { // si es impar participa solo en
el secundario excepto en el primer partido
          this.matchesSecondary[_j].push(this.matchesJourneys[_j][_m]);
        } else if ( _j === 0) {
          this.matchesPrincipal[ j].push(this.matchesJourneys[ j][ m]);
        }
      }
    }
    this.results = true;
    this.ionicService.removeLoading();
  }

  showMore() {
    this.show === true ? this.show = false : this.show = true;
  }
}
```

```
<ion-header>
  <ion-navbar color="primary">
    <button ion-button menuToggle>
      <ion-icon name="menu"></ion-icon>
    </button>
  <ion-title>{{ 'COMPETITIONS.CALENDAR' | translate }}</ion-title>
</ion-navbar>
</ion-header>

<ion-content padding>
    <ion-refresher (ionRefresh)="getJourneys($event)">
      <ion-refresher-content></ion-refresher-content>
    </ion-refresher>
  <div *ngIf="results">
        <h4>{{ 'COMPETITIONS.CALENDAR OF2' | translate }} {{ 'TENNIS.TITLE' | translate
}}: {{ competition.name }}</h4>
            <ion-card class="journey-card" *ngFor="let date of dates; let i=index;">
              <ion-card-content>
                  <p><u><b>{{   'COMPETITIONS.JOURNEY'   |   translate   }}   {{i   +
1}}</b></u></p>
                  <p   class="date"><i>{{   'COMPETITIONS.DATE'   |   translate   }}:
{{date}}</i></p>
               <table *ngIf="matchesPrincipal[i].length !== 0">
                  <caption>
                   <p  *ngIf="i  !==  numJourneys  -  1"><b>{{  'TOURNAMENTS.PRINCIPAL'  |
translate }}</b></p>
                    <p  *ngIf="i  ===  numJourneys  -  1"><b>{{  'TOURNAMENTS.FINAL'  |
translate }}</b></p>
                  </caption>
                  <thead>
                    <tr class="tableHeader">
```

```
                    <th *ngIf="competition.mode === 'Individual'" class="player">{{
'COMPETITIONS.PLAYER' | translate }} 1</th>
                    <th *ngIf="competition.mode === 'Individual'" class="player">{{
'COMPETITIONS.PLAYER' | translate }} 2</th>
                    <th  *ngIf="competition.mode  ===  'Equipos'"  class="player">{{
'COMPETITIONS.TEAM' | translate }} 1</th>
                    <th  *ngIf="competition.mode  ===  'Equipos'"  class="player">{{
'COMPETITIONS.TEAM' | translate }} 2</th>
                    <th class="result">{{ 'COMPETITIONS.WINNER' | translate }}</th>
                  </tr>
                </thead>
                <tbody>
                  <tr *ngFor="let match of matchesPrincipal[i]">
                    <td class="player">{{match.namePlayerOne}}</td>
                    <td class="player">{{match.namePlayerTwo}}</td>
                    <td class="result">{{match.result}}</td>
                  </tr>
                </tbody>
              </table>
              <div class="thead-tbody" *ngIf="matchesSecondary[i].length !== 0">
              <p><b>{{ 'TOURNAMENTS.SECONDARY' | translate }}</b></p>
              <table>
                <thead>
                  <tr class="tableHeader">
                    <th *ngIf="competition.mode === 'Individual'" class="player">{{
'COMPETITIONS.PLAYER' | translate }} 1</th>
                    <th *ngIf="competition.mode === 'Individual'" class="player">{{
'COMPETITIONS.PLAYER' | translate }} 2</th>
                    <th  *ngIf="competition.mode  ===  'Equipos'"  class="player">{{
'COMPETITIONS.TEAM' | translate }} 1</th>
                    <th  *ngIf="competition.mode  ===  'Equipos'"  class="player">{{
'COMPETITIONS.TEAM' | translate }} 2</th>
                    <th class="result">{{ 'COMPETITIONS.WINNER' | translate }}</th>
                  </tr>
                </thead>
                <tbody>
                  <tr *ngFor="let match2 of matchesSecondary[i]">
                    <td class="player">{{match2.namePlayerOne}}</td>
                    <td class="player">{{match2.namePlayerTwo}}</td>
                    <td class="result">{{match2.result}}</td>
                  </tr>
                </tbody>
              </table>
            </div>
            </ion-card-content>
          </ion-card>
        </div>
        <div class="foot">
            <ion-card class="foot">
          * {{ 'TENNIS.GHOST1' | translate }}
          {{ 'TENNIS.GHOST2' | translate }}.
          </ion-card>
        </div>
        <div *ngIf="datesNoMatches">
          <div *ngIf="datesNoMatches.length !==0">
            <h6 class="foot2" *ngIf="!show">{{ 'TENNIS.MORE JOURNEY1' | translate }}.
{{ 'TENNIS.MORE JOURNEY2' | translate }}:
            </h6>
            <h6  class="foot2"  *ngIf="show">{{  'TENNIS.LESS_JOURNEY'  |  translate
}}:</h6>
            <button ion-button class="morebtn" icon-only color="warn" *ngIf="!show"
(click)="showMore()">
                <ion-icon name="add"></ion-icon>
            </button>
            <button ion-button class="morebtn" icon-only color="warn" *ngIf="show"
(click)="showMore()">
                <ion-icon name="remove"></ion-icon>
            </button>
            <div *ngIf="show" class="cardsNoMatch-div">
            <ion-card    class="dateNoMatch-card"    *ngFor="let    dateNoMatch    of
datesNoMatches; let i=index;">
                <ion-card-header>
                    <p><u><b>{{     'COMPETITIONS.JOURNEY'     |     translate     }}
{{dateNoMatch.number}}</b></u></p>
                  <p    class="date">{{    'COMPETITIONS.DATE'    |    translate    }}:
{{dateNoMatch.date}}</p>
                </ion-card-header>
```

```
                <ion-card-content>
                  <ion-list role="list">
                     <ion-item    role="listitem"    *ngFor="let    tournament    of
tournaments[i]">- {{tournament}}</ion-item>
                  </ion-list>
                </ion-card-content>
              </ion-card>
        </div>
      </div>
      </div>
</ion-content>
```

9.9.8. Equipos de la liga y del torneo de tenis (compartido)

```
export class TeamsPage {

  public competitionId: string;
  public teams: Array<Team>;
  public allStudents: Array<Array<Student>>;

  constructor(public navCtrl: NavController,
    public navParams: NavParams,
    public ionicService: IonicService,
    public utilsService: UtilsService,
    public groupService: GroupService,
    public competitionService: CompetitionService,
    public translateService: TranslateService) {}

  ionViewDidLoad() {
    this.ionicService.showLoading(this.translateService.instant('APP.WAIT'));
    this.competitionId = this.navParams.get('competitionId');
    this.getTeams();
  }

  /**
   * This method returns the teams list of the current competition
   */
  private getTeams(): void {
    this.competitionService.getTeamsCompetition(this.competitionId)
    .subscribe((teams => {
        this.teams = teams,
        this.getStudents();
      }),
      (error => {
        this.ionicService.removeLoading();
        this.ionicService.showAlert(this.translateService.instant('APP.ERROR'),
error);
      }));
  }
  /**
   * This method returns the students list of each team
   */
  private getStudents(): void {
    let countTeams = 0;
    this.allStudents = [];
    for (let  t = 0;  t < this.teams.length;  t++) {
    this.competitionService.getStudentsTeam(+this.teams[ t].id)
    .subscribe( (students => {
      this.allStudents[_t] = students;
      this.teams[_t].numPlayers = students.length;
      countTeams = countTeams + 1;
      countTeams === this.teams.length ? this.ionicService.removeLoading() : null;
    }),
    (error => {
      this.ionicService.removeLoading();
      this.ionicService.showAlert(this.translateService.instant('APP.ERROR'), error);
    }));
    }
  }
}
```

```
<ion-header>
    <ion-navbar color="primary">
        <button ion-button menuToggle>
          <ion-icon name="menu"></ion-icon>
        </button>
        <ion-title>{{ 'COMPETITIONS.TEAMS' | translate }}</ion-title>
      </ion-navbar>
</ion-header>


<ion-content padding>
  <ion-card class="team-card" *ngFor="let team of teams; let i=index;">
    <ion-card-header>
      <h1><strong>{{ 'COMPETITIONS.TEAM' | translate }} {{i + 1}}: {{team.name |
uppercase}}</strong></h1>
      <h2>{{team.numPlayers}} {{ 'COMPETITIONS.PLAYERS' | translate }}</h2>
    </ion-card-header>
        <ion-list *ngFor="let student of allStudents[i]">
          <button ion-item>
            <ion-icon name="person" item-start></ion-icon>
            {{student.name}} {{student.surname}}
          </button>
        </ion-list>
  </ion-card>
</ion-content>
```

9.10. Cuestionario sobre el generador de competiciones de Classpip

Generador de competiciones de Classpip

Cuestionario anónimo de calidad y mejora para la aplicación Classpip y su nueva herramienta generadora de competiciones.

*Obligatorio

Videotutorial sobre la sección de competiciones de Classpip

¿A qué colectivo pertenece? *

- ◯ Alumno/a
- ◯ Docente
- ◯ Otro

¿A qué grupo de edad pertenece? *

- ◯ Menor de 14 años
- ◯ Entre 14 y 18 años
- ◯ Entre 19 y 25 años
- ◯ Mayor de 26 años

Figura 9.61. Cuestionario (Parte 1)

¿Qué le parece la aplicación en general? *

- ◯ 1 - Mala
- ◯ 2 - Regular
- ◯ 3 - Buena
- ◯ 4 -Muy buena

¿Qué le parece la estructura de la aplicación? *

- ◯ 1 - Mala
- ◯ 2 - Regular
- ◯ 3 - Buena
- ◯ 4 - Muy buena

¿Qué le parece la facilidad de uso de la aplicación? *

- ◯ 1 - Difícil
- ◯ 2 - Regular
- ◯ 3 - Fácil
- ◯ 4 - Muy fácil

¿Qué le parece el diseño gráfico de la aplicación? *

- ◯ 1- Malo
- ◯ 2 - Regular
- ◯ 3 - Bueno
- ◯ 4 - Muy bueno

Figura 9.62. Cuestionario (Parte 2)

¿Qué le parece el tipo y tamaño de letra de la aplicación? *

- ◯ 1 - Malo
- ◯ 2 - Regular
- ◯ 3 - Bueno
- ◯ 4 - Muy bueno

¿Cómo de recomendable calificaría la aplicación? *

- ◯ 1 - Muy poco recomendable
- ◯ 2 - Poco recomendable
- ◯ 3 - Recomendable
- ◯ 4 - Muy recomendable

Si tuviese la ocasión, ¿utilizaría la aplicación para impartir conocimiento como docente? *

- ◯ Sí
- ◯ No

¿Qué mejoras introducirías en la aplicación?

Tu respuesta

¿Qué nuevas funcionalidades introducirías en cada competición?

Tu respuesta

ENVIAR

Figura 9.63. Cuestionario (Parte 3)