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

TRABAJO FINAL DE GRADO

TÍTULO DEL TFG: Mejora de la base de datos de una herramienta de gamificación e implementación de una conexión con una plataforma de aprendizaje

TITULACIÓN: Grado en Ingeniería de Telemática

AUTOR: Andrés Fernando Caicedo Uvidia

DIRECTOR: Miguel Valero

FECHA: 15 de septiembre del 2020

Título: Mejora de la base de datos de una herramienta de gamificación e implementación de una conexión con una plataforma de aprendizaje

Autor: Andrés Fernando Caicedo Uvidia

Director: Miguel Valero

Fecha: 15 de septiembre del 2020

Resumen

En este proyecto partimos de la gamificación [1] como técnica de mejora del aprendizaje. La gamificación consiste en hacer uso de las mecánicas de los juegos dentro del ámbito educativo y profesional con el objetivo de mejorar los resultados, conseguir absorber mejor los conocimientos o mejorar habilidades.

Me centraré en el proyecto de Classpip [2] como herramienta de gamificación ya que me permite introducir mecánicas de puntos, cuestionarios, avatares, etc. Mis contribuciones principales al proyecto son: mejorar la base de datos y permitir que la información de Classpip sea accesible desde otras plataformas de aprendizaje, como por ejemplo Moodle [3]. Es por ello que se requiere de la implementación de una solución que permita conectar Classpip con Moodle mediante el protocolo LTI [4].

Como resultado, el proyecto se divide en dos secciones principales para cada uno de los objetivos. En primer lugar, me he centrado en la investigación e implementación de una base de datos, adecuada a la herramienta de Classpip. En segundo lugar, he buscado y creado un prototipo sobre cómo conectar, mediante el protocolo LTI, la aplicación de Classpip con una plataforma de aprendizaje como Moodle.

En resumidas cuentas, el primer objetivo de este proyecto es el de poder respaldar todos los datos de la herramienta de gamificación Classpip. De esta forma, se consigue custodiar de manera segura, rápida y escalable, toda la información necesaria para el uso de Classpip. Esta parte del proyecto surgió tras revisar la forma y el estado en el que se guardaban los datos de la aplicación. Pude observar la necesidad de mejorar el método utilizado y, de esta manera, encontrar una solución que permitiese prescindir del uso de archivos como respaldo para la información. Al mismo tiempo, aumentamos la fiabilidad y reducimos los riesgos asociados a los ficheros cómo método de archivado de datos. Como resultado, hemos conseguido migrar los datos de Classpip a la base de datos no relacional mongoDB [5], lo cual implica grandes mejores y más facilidad de manejo de datos.

El segundo objetivo es poder utilizar Classpip como proveedor de contenido mediante la creación de una conexión con el protocolo LTI. Esto nos permite hacer uso de los componentes de una aplicación externa dentro de una

plataforma de aprendizaje. Por lo consiguiente, nos ha permitido utilizar los elementos que podemos encontrar en Classpip dentro de Moodle. Es así como establecemos una relación entre una plataforma y un proveedor de contenido externo, abriendo camino a posibilidades muy interesantes.

Cada uno de los objetivos planteados da como resultado una serie de productos. Los dos primeros productos son guías que ayudarán a aquellas personas que quieran implementar dos tipos de base de datos, mongoDB y MySQL [6], en un proyecto como Classpip. El resto de productos pertenecen al segundo objetivo. Estos productos se dividen en varias guías que explican la manera de implementar las funciones necesarias para hacer uso de los diferentes juegos que componen Classpip.

Title: Improvement of the database of a gamification tool and implementation of a connection with a learning platform

Author: Andrés Fernando Caicedo Uvidia

Director: Miguel Valero

Date: 15th September 2020

Overview

In this project we start from gamification [1] as a learning improvement technique. Gamification consists of making use of the mechanics of games within the educational and professional sphere with the aim of improving results, better absorbing knowledge or improving skills.

I will focus on the Classpip [2] project as a gamification tool since it allows me to introduce point mechanics, quizzes, avatars, etc. My main contributions to the project are: improving the database and making Classpip information accessible from other learning platforms, such as Moodle [3]. That is why it is required to implement a solution that allows connecting Classpip with Moodle through the LTI protocol [4].

As a result, the project is divided into two main sections for each of the objectives. In the first place, I have focused on the research and implementation of a database, suitable for the Classpip tool. Second, I have searched and created a prototype on how to connect, using the LTI protocol, the Classpip application with a learning platform like Moodle.

In short, the first objective of this project is to be able to back up all the data from the Classpip gamification tool. In this way, it is possible to safeguard in a safe, fast and scalable way, all the information necessary for the use of Classpip. This part of the project came about after reviewing the way and the state in which the application data was saved. I was able to observe the need to improve the method used and, in this way, find a solution that would make it possible to dispense with the use of files as backup for the information. At the same time, we increase reliability and reduce risks associated with files as a data archiving method. As a result, we have managed to migrate the Classpip data to the non-relational mongoDB [5] database, which means great improvements and easier data handling.

The second goal is to be able to use Classpip as a content provider by creating a connection with the LTI protocol. This allows us to make use of the components of an external application within a learning platform. Therefore, it has allowed us to use the elements that we can find in Classpip within Moodle.

This is how we establish a relationship between a platform and an external content provider, opening the way to very interesting possibilities.

Each of the proposed objectives results in a series of products. The first two products are guides that will help those who want to implement two types of databases, mongoDB and MySQL [6], in a project like Classpip. The rest of the products belong to the second objective. These products are divided into several guides that explain how to implement the necessary functions to make use of the different games that make up Classpip.

ÍNDICE

INTRODUCCIÓN

Estamos en un mundo rodeado de información. En este período de tiempo los datos son el centro de casi todo. Aquel que los controla, controla el resto. Es por eso que es primordial el poder estar seguro de que están bien guardados, tanto para el usuario como para el proveedor. Asimismo, pienso que Classpip necesitaba una renovación en la forma mediante la cual se almacenaban los datos. Esto cobra mucho más sentido si tenemos en cuenta que se necesitan custodiar de una manera segura y eficaz.

Classpip utiliza ciertas mecánicas, dinámicas y estéticas de videojuegos que nos ofrecen una solución al problema de la poca disposición a aprender por las enseñanzas tradicionales. Este nuevo entorno educativo ayuda a despertar ese interés, muchas veces escondido, del usuario por aprender, consiguiendo un mejor rendimiento académico. En simples palabras, un juego que te ayuda a aprender. Classpip lo implementa mediante la gamificación para crear un entorno y poder, de esta forma, establecer un entorno gamificado.

La gamificación es un recurso de aprendizaje que nos permite transferir la mecánica, dinámica y estética de los juegos. De esta manera, logra optimizar los resultados académico-profesionales, ayuda a interiorizar mejor los conocimientos y genera experiencias positivas. Todo esto lo alcanza mediante el uso de métodos más entretenidos, que consiguen despertar la motivación, el compromiso y el espíritu de superación de cada usuario.

Así mismo, para mejorar la experiencia del usuario, surgen ciertas herramientas que permiten el uso de aplicaciones ajenas a una plataforma o entorno de aprendizaje. Este tipo de herramientas hacen uso del protocolo LTI (del inglés Learning Tools Interoperability o Interoperabilidad entre Herramientas de Aprendizaje). Mediante el uso de dichas herramientas, se consigue integrar una aplicación cualquiera dentro de la plataforma educativa que se escoja. Es así como se podrá hacer uso de sus componentes sin la necesidad de salir del entorno educativo. Esto, a su vez, mejora el rendimiento al no tener que cambiar de página para acceder a otro contenido.

En relación con los conceptos expuestos, este proyecto consta de dos secciones bien diferenciadas. La primera sección se centra en el desarrollo e implementación de una base de datos para Classpip. La segunda parte se enfoca en cómo implementar una herramienta que haga uso del protocolo LTI para conectar Moodle con Classpip.

Del mismo modo que este proyecto se divide en dos secciones, los objetivos y los productos, derivados de cada una de ellas, también.

Mientras que en mi primer objetivo me centro en la creación de la base de datos para Classpip, en el segundo, me focalizo en implementar Classpip cómo herramienta externa mediante el uso de una librería LTI. En mi caso, me centraré en la librería llamada LTIJS [7].

El primer objetivo da como resultado dos productos: una guía para adaptar una aplicación a mongoDB y otra para MySQL. Del segundo objetivo obtengo dos productos: una guía para implementar paso a paso la librería LTIJS y un prototipo de demostración de dicha implementación.

Dentro de este marco he divido la estructura del proyecto de la siguiente manera:

Introducción al proyecto de Classpip y su arquitectura, en el Capítulo 1.

Explicación de qué son las bases de datos y los diferentes tipos y la selección de la base de datos juntamente con su implementación, se encuentra en el Capítulo 2.

Todo lo referente a qué es LTIJS, como se implementa y los retos enfrentados, definido en el Capítulo 3.

La implementación de la librería LTIJS, usando Classpip y Moodle como base, explicada en el Capítulo 4.

Conclusiones, tanto personales como técnicas, al igual que futuras mejoras, descritas en el Capítulo 5.

Y finalmente tenemos los Anexos en los que se encuentran las guías mencionadas con anterioridad: implementar mongoDB y MySQL y llevar a cabo la implementación de LTIJS.

Cómo última reflexión, para llevar a cabo este proyecto he tenido que gestionar múltiples tareas de diferentes ámbitos: una parte de base de datos, otra de LTIJS, búsqueda de información, creación del documento y administrar el tiempo para el proyecto, mientras estudiaba y/o trabajaba a tiempo completo. Por consiguiente, he ganado experiencia que puedo incorporar dentro de mi carrera profesional, lo cual encuentro que forma parte del proceso de realización del TFG.

En resumidas cuentas, todo lo explicado me aporta un valor inestimable que me ayudará a crecer profesionalmente.

Y sin más preámbulos es el momento de explicar qué es Classpip.

CAPÍTULO 1. CLASSPIP

Classpip es una herramienta de gamificación desarrollada por estudiantes de la Escuela de Ingeniería de Telecomunicación y Aeronáutica de Castelldefels (EETAC), de la Universidad Politécnica de Cataluña (UPC). El propósito es el de aumentar las ganas y el interés de los estudiantes por aprender, mediante el uso de dinámicas y mecánicas de juego junto con nuevas tecnologías.

La aplicación fue iniciada en el año 2016 por el Departamento de Arquitectura de Computadores y en su desarrollo han participado más de 20 estudiantes haciendo aportaciones a través de sus TFG o TFM.

Actualmente se dispone de una gran variedad de funciones y juegos, así como una aplicación móvil para los profesores y alumnos.

En este capítulo se describe brevemente las funcionalidades de la aplicación, y su arquitectura. También se describe brevemente en qué elementos de la arquitectura ha incidido mi TFG y qué tecnologías he usado para desarrollarlo.

He dividido la explicación en los siguientes apartados:

1.1 Funcionalidades de Classpip.

1.2 Arquitectura de Classpip.

1.3 Elementos afectados por mi proyecto y tecnologías utilizadas.

1.1. Funcionalidades de Classpip

Classpip es una herramienta que permite a los profesores inscribir y gestionar los grupos de alumnos a los que enseñan. Pueden crear juegos individuales o por equipos (seleccionados por el profesor).

El profesor puede personalizar los juegos usando la creación de los siguientes materiales para poder jugarlos:

- **Crear puntos e insignias:** los profesores pueden recompensar a los estudiantes en el juego de puntos.

- **Crear colecciones y sus respectivos cromos:** los profesores los pueden distribuir a los estudiantes en el juego de colección.

- **Crear privilegios:** son dados a los alumnos por su profesor para que puedan tener acceso a los componentes del juego de avatar.

Además, la aplicación también se puede utilizar para pasar lista y controlar la asistencia de los estudiantes, así como también premiar la puntualidad.

1.1.1. Juego de Puntos

Este juego consiste en asignar puntos a alumnos / equipos premiando determinadas actitudes y logros (comportamiento, puntualidad, buenas notas en los exámenes, etc.). El juego se compone niveles creados por el profesor al generar el juego. En estos niveles se otorgan ciertos privilegios a los participantes que los superen.

El objetivo es ser el primero en la clasificación, alcanzando cada nivel para obtener su recompensa correspondiente.

1.1.2. Juego de Competición

En la modalidad de "competición", se puede elegir entre "Liga" y "Torneo". Dependiendo de la opción seleccionada, el objetivo será diferente.

- **Liga:** El objetivo es estar lo más alto posible en la clasificación. Todos los días, se produce un enfrentamiento entre dos participantes. El ganador de la jornada se decidirá según los criterios que determine el profesor en esa fecha concreta. El ganador recibirá puntos de clasificación en la liga, con posibilidad de obtener otro tipo de recompensas (como cromos) de otros juegos activos.

- **Torneo:** se juegan partidos de eliminación hasta que solo quede un jugador. Se especifican los criterios para ganar y las recompensas. En esta modalidad, una vez eliminado, no puedes obtener más recompensas.

1.1.3. Juego de Colección

Este modo de "Colección" se basa en otorgar a alumnos / equipos con cromos. Estos cromos pertenecen a una colección previamente creada y seleccionada para el juego. El profesor asigna los cromos a cada alumno de manera deliberada o aleatoria. El objetivo final es conseguir el álbum con todos los cromos.

A diferencia de los juegos de puntos, no se especifican objetivos claros al crear juegos para ganar recompensas.

1.1.4. Juego de Avatar

En este juego los alumnos disponen de un avatar. Este solo se puede personalizar ganando privilegios. Cada avatar pertenece a una familia previamente creada por el profesor, el cual es el encargado de otorgar privilegios en función de si el alumno cumple o no los objetivos especificados. El objetivo final es poder personalizar libremente tu avatar, poder ver los avatares de tus compañeros y ser capaz de usar tu voz para el avatar.

Al igual que el juego de colección, no se especifican objetivos claros al crear juegos para ganar recompensas.

1.1.5. Juego de Cuestionario

Este juego se centra en que el profesor crea un cuestionario y agrega las preguntas que considere necesarias. Por cada pregunta hay 4 respuestas posibles siendo solo 1 la correcta.

El alumno contesta cada pregunta del juego de Cuestionario que tenga asignado y, en función de su puntuación, puede recibir recompensas como cromos o privilegios de otros juegos.

1.2. Arquitectura de Classpip

Cada uno de los 5 proyectos de los que se compone Classpip tiene sus particularidades. Tanto el proyecto de Classpip-mobile (para el profesor) como el del alumno parten de las mismas bases, por lo tanto, los explicaré en conjunto.

Los proyectos se dividen en los siguientes:

- **Classpip-dashboard**: Este proyecto permite tener acceso a la aplicación desde un navegador web, preferiblemente desde un ordenador. Está enfocado para que solo el profesor pueda utilizarlo. Permite configurar todas las características para gestionar: juegos, alumnos, grupos, asignación de puntos, etc.

  Se basa en las siguientes tecnologías: Node.js [8], Express [9], Angular [10], Angular Material [11] y Bootstrap [12].

- **Classpip-mobile (profesor y alumno):** Permite el acceso a parte del contenido de la aplicación desde un dispositivo móvil. Está basado en el *framework* de *Ionic 2* [13] junto a otras herramientas como *Angular* (para hacerlo dinámico) y *Angular Material* (para el diseño de la interfaz).

  Hace uso de *Apache Cordova* [14] lo que permite tener acceso a herramientas del dispositivo móvil como el lector de huella, la cámara, el GPS, etc.

- **Classpip-services:** Hace uso de API REST [15] para conectarse a la base de datos. Utiliza *Strong Loop* [16] como *endpoint* [17], el cual a su vez está basado en *express* y *node.js*. Esta tecnología permite acceder a la información de la de datos mediante direcciones web *HTML* [18].
  Es por ello que tanto el Dashboard como el Mobile tienen acceso a él, a través del Servidor.

Como base de datos se hace uso de un fichero .json con todos los datos en él.

- **Servidor:** Basado en *node.js* y *express*. Pone en marcha la aplicación de Classpip y se encarga de gestionar las notificaciones entre el Dashboard y el móvil del alumno (por ejemplo, cuando un alumno responde a un cuestionario se notifica al Dashboard, o cuando el profesor asigna nuevos cromos a un alumno, se notifica al móvil del alumno).

La relación que se establece entre los 5 proyectos es la que podemos ver en la siguiente **Fig 1.3.1**.

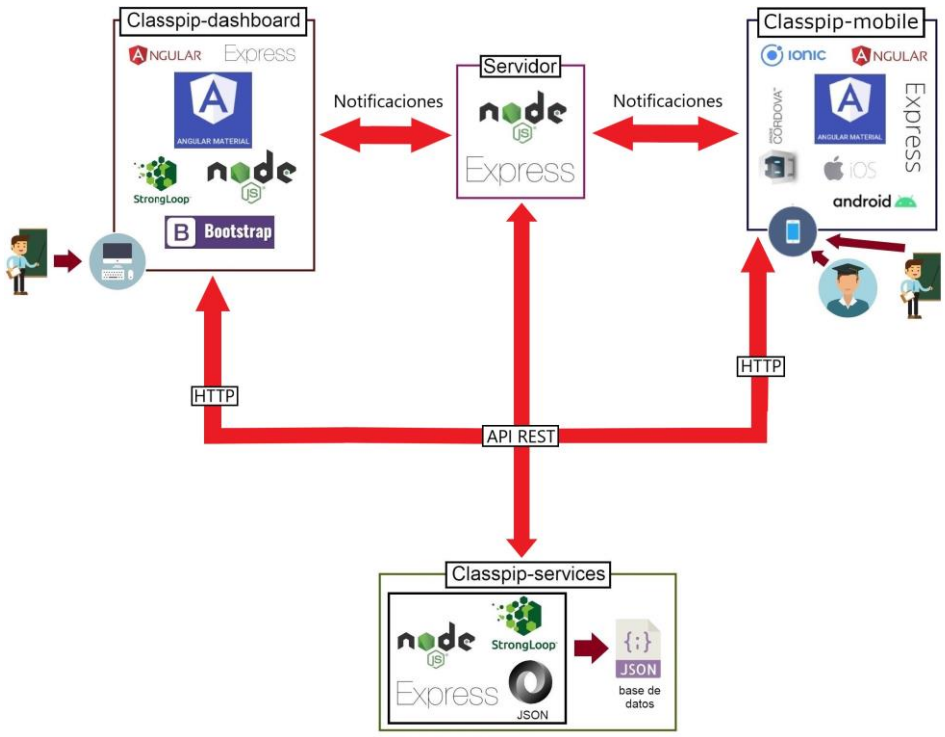

Fig 1.3.1 Relación entre los proyectos de Classpip

Tal y como se aprecia en la **Fig 1.3.1**, el Dashboard, el Mobile (profesor y alumno) y el Servidor acceden a la API REST del Services. En los casos del Dashboard y el Mobile, el acceso se produce mediante peticiones HTTP. El Servidor se encarga de gestionar las notificaciones entre el Dashboard y el Mobile.

Para realizar las modificaciones pertinentes, en el caso de la base de datos, haré los cambios en el proyecto Classpip-services y, con LTIJS, los cambios los realizaré en el Servidor.

1.3. Elementos afectados por mi proyecto y tecnologías usadas

Para mi proyecto me he centrado en dos proyectos de los cinco proyectos: Classpip Services y Servidor. Para la parte de la base de datos me centré en el Classpip Services para poder gestionar el guardado de los datos. En el caso del apartado de LTIJS, hice uso del Servidor ya que es el encargado de gestionar los servicios de la aplicación. Esto permite incorporar la conectividad LTI.

En este proyecto he partido de una base tecnológica ya establecida:

- **Strongloop LoopBack 3:** permite gestionar la API REST (generando modelos, relaciones entre modelos, etc.).

- **Visual Studio Code** [19]**:** editor de código.

- **GitHub** [20]**:** para facilitar las contribuciones de cada miembro del equipo. GitHub permite guardar tu proyecto en la nube y juntar el progreso de cada alumno.

Por otra parte, la tecnología implementada para llevar a cabo los objetivos de mi proyecto de base de datos ha sido la siguiente:

- **Loopback-connector-mongodb** [21]**:** para realizar la conexión entre el proyecto y la base de datos.

- **Loopback-connector-mysql** [22]**:** para realizar la conexión entre el proyecto y la base de datos.

Y para el caso del proyecto de LTI:

- **Moodle:** uno de los muchos tipos de plataforma de aprendizaje. Necesaria para poder hacer la implementación.

- **LTIJS:** librería para implementar el protocolo LTI (v1.3) en nuestro proyecto.

- **Dotenv:** para poder cargar variables de entorno desde un fichero .env (requerido para facilitar el uso de LTIJS).

- **Babel-eslint:** para soportar las características que no tenemos en ESLint y poder parsear correctamente el valor de =>.

- **Mongodb:** base de datos soportada por defecto en LTI.

- **EJS:** Motor de modelos para generar páginas HTML y tener acceso a variables del servidor.

Tras las modificaciones realizadas, las relaciones entre los proyectos quedan tal y como se muestra en la **Fig. 1.3.1**.

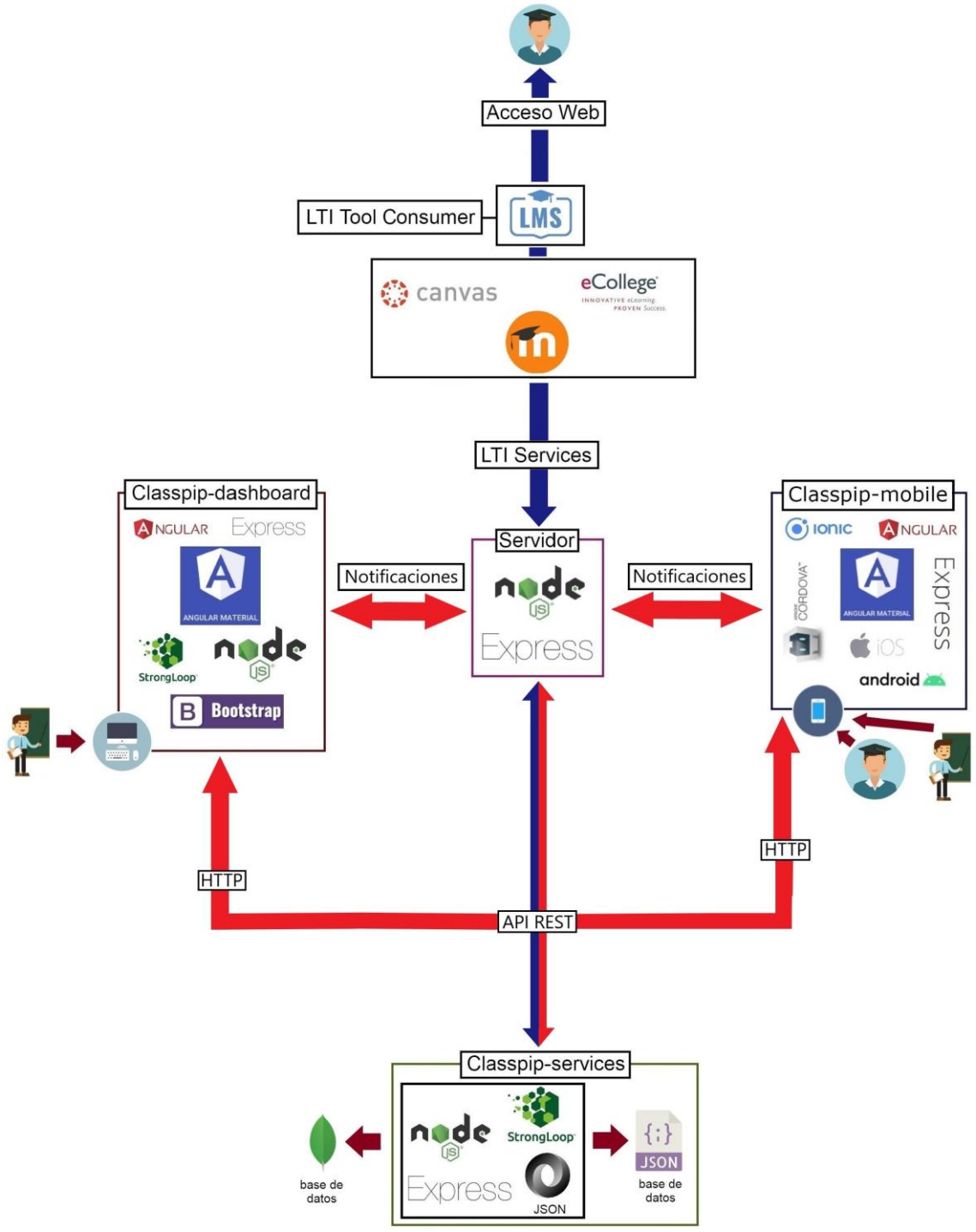

Fig. 1.3.1 Relación entre los proyectos de Classpip tras las implementaciones

Tal y como se ve en la **Fig 1.3.1**, se añade una nueva relación entre un alumno, el Servidor y el Classpip-services (marcado en azul). Esta relación se basa en el uso de una librería de LTI para realizar la conexión entre la plataforma de Moodle y la herramienta de Classpip.

Con los cambios realizadas las relaciones entre los 5 proyectos que forman Classpip no ha cambiado. Simplemente se ha añadido una nueva relación que solo afecta al Servidor ya que es el que contiene toda la implementación de LTIJS. Con este proyecto se consigue hacer uso de Classpip en otros ámbitos externos a la propia aplicación. También se consigue mejorar de manera sustancial la forma de guardar los datos.

CAPÍTULO 2. BASES DE DATOS

En este apartado explicaremos los diferentes tipos de base de datos que existen y sus limitaciones, veremos la estructura actual de la base de datos de Classpip, veremos el proceso para escoger la base de datos y cómo se implementa y acabaremos con unas conclusiones.

Una base de datos es un conjunto de datos, pertenecientes al mismo contexto, que se almacenan de manera sistemática para utilizarlos a posteriori.

2.1. Clasificación de base de datos

Dependiendo de la función o utilidad, las bases de datos se clasifican en diversos tipos. En este proyecto nos centraremos en la clasificación según los modelos de administración de datos que usen, específicamente en dos tipos: las relacionales y las documentales. Esto significa que dependiendo de la forma en la que guarden los datos o los métodos que utilicen para almacenar y recuperar dichos datos, tendremos un tipo u otro.

Los dos tipos de base de datos que analizaremos son:

- **Bases de datos relacionales:** se basa en tablas y en las relaciones que tienen entre ellas. El lenguaje que predomina es el conocido como SQL [23] y rige la integridad de los datos. Esto significa que se hace uso de ciertas restricciones, como el “Not NULL”, con tal de aplicar la integridad de la información. Este tipo de base de datos usa transacciones, las cuales son formadas por una o diversas instrucciones SQL ejecutadas en secuencia. Algo importante a mencionar es que todas las transacciones deben ser conformes a ACID [24] con tal de garantizar la integridad de los datos.

  Con atomicidad nos referimos a que la transacción es un todo, o se ejecuta correctamente o si una parte falla, toda ella se invalida. La consistencia implica que los datos escritos cumplan con las restricciones o reglas que se hayan definido previamente. El aislamiento nos permite que cada transacción sea independiente. Y, por último, la durabilidad requiere que aquellos cambios hechos sean permanentes tras completar la transacción.

  Como punto positivo, este tipo de base de datos, son muy útiles si los datos no requieren modificaciones constantes. Un ejemplo seria MySQL.

- **Base de datos documentales:** basadas en registros y datos, se construyen usando el lenguaje NoSQL y utilizando ficheros (como los JSON [25] mencionados anteriormente).

Como ventaja, son muy útiles cuando requieres de flexibilidad con tus datos. Además, aportan gran rapidez a la hora de consultar y hacer modificaciones de los datos, tienen gran escalabilidad y proporcionan un gran rendimiento en cuanto a rapidez. Un ejemplo es mongoDB.

2.2. Base de datos de Classpip

Classpip utiliza como base de datos principal un fichero *.json*, en el cual incluyen todos los datos necesarios pertenecientes a los usuarios (nombres, contraseñas, juegos, grupos, etc.). Esta base de datos es del tipo documental.

Un fichero *.json* es simplemente un archivo que contiene datos en un formato de texto específico (JSON). Un ejemplo es el siguiente:

```
{"dept":8,"nombredept":"Ventas","director":"juan","empleados":[{"nombre":"Pedr
o","apellido":"Fernandez"},{"nombre":"Jacinto","apellido":"Benavente"} ]}
```

Las ventajas de este tipo de ficheros son las siguientes:

- Es autodescriptivo y fácil de entender.
- Se parsea rápido.
- Facilidad de manejo: se puede modificar y trasladar a otro lugar de manera sencilla.
- No ocupa mucho espacio.

Los inconvenientes son:

- No es escalable cuando los datos aumentan sustancialmente.
- No es eficaz en cuanto a rendimiento ya que, si no se puede guardar el fichero dentro de la memoria RAM [26], el rendimiento decrece mucho.
- No tienes una base de datos tan ordenada ya que todo se encuentra en un mismo fichero.
- No hay verificación del contenido: aquel que escriba el fichero debe controlar que la información introducida sea correcta.
- El rendimiento disminuye conforme más datos hay.

Es debido a estas razones que he buscado, entre diferentes opciones de bases de datos, la que mejor encajaba con este tipo de aplicación.

2.3. Selección de base de datos

Ahora nos centraremos en explicar el proceso de selección de la base de datos.

Primero de todo, explicaré qué necesidades hay en la aplicación de Classpip, compararé diferentes tipos de base de datos actuales y finalmente mostraré el rendimiento de diferentes bases de datos, a modo de comparación, para finalmente escoger una base de datos.

2.3.1. Necesidades de la base de datos de Classpip

Classpip necesita una base de datos que supla las deficiencias del fichero *.json* y que, al mismo tiempo, cómo mínimo aporte ventajas iguales o parecidas.

Es por ello que necesitamos una base de datos que sea fácil de utilizar, que sea escalable y flexible, cuyo rendimiento no disminuya en gran medida, que nos de algún tipo de feedback si introducimos datos incorrectos o nos falta algo y que sea flexible. Este último punto es importante debido a que, con el tiempo, irán surgiendo modificaciones o nuevos datos que gestionar, y si la base de datos es muy rígida, la implementación de los cambios no será tan sencilla.

También tengo en cuenta que la solución plantada es en el ámbito local antes que en la nube.

2.3.2. Comparación de bases de datos

La **Tabla 2.3.2.1** compara los diferentes tipos de bases de datos. Podemos encontrar tanto versiones gratuitas como de pago.

Tabla 2.3.2.1. Comparación entre bases de datos

	MySQL	mongoDB	Oracle	MS SQL
Tipo	Relacional	No relacional	Multi modelo	Relacional
Lanzamiento	1995	2009	1979	1989
Esquema	Rígido	Flexible	Rígido	Rígido
Almacenamiento	Tabla	Fichero BSON	Diferentes tipos	Tabla
Tipo de dato	Estructurado	No estruct.	Estruct. y no estruct.	Estructurado
Tipo operación	Transacción entre tablas	Sin esquema	Mixto	Transacción entre tablas
Licencia	Open Source	Open Source	Comercial	Comercial

Como podemos observar a partir de los datos de la **Tabla 2.3.2.1**, podemos descartar las dos bases de datos con licencia comercial (de pago). Nos quedamos con MySQL y con mongoDB. Ya podemos observar como mongoDB es la base de datos que ofrece una estructura flexible, lo cual encaja con uno de nuestros requisitos. Así mismo, no depende de transacciones por lo que implica más rapidez a la hora de consultar datos.

2.3.3. Rendimiento de las bases de datos

Es importante comparar el rendimiento a la hora de acceder, modificar o borrar datos dependiendo de qué base de datos se utilice.

Utilizando la herramienta *Postman* [27], he medido los diferentes tiempos al realizar diferentes consultas usando 3 tipos diferentes de base de datos: mongoDB, el fichero .json y MySQL.

Los resultados que he obtenido son los que se pueden ver en la **Tabla 2.3.3.1**.

Tabla 2.3.3.1. Tiempo (ms) de las consultas según la base de datos utilizada

Consulta	mongoDB	.json	MySQL
Login	509	80	518
Grupos	535	80	525
Puntos	511	79	519
Colección	523	80	522
Liga	516	81	517
FormulaUno	511	79	515
Avatar	515	80	513
Cuestionario	524	80	519
Añadir alumno	519	81	550
Borrar alumno	521	105	520

Cabe destacar que los resultados obtenidos forman parte de una media de una serie de medidas tomadas. Realicé, por cada consulta, 10 pruebas para encontrar un poco de estabilidad en los datos.

De la **Tabla 2.3.3.1** se puede ver que el fichero *.json* tiene los tiempos de consulta más bajos. Esto, claro está, si estamos trabajando con una cantidad de información bastante reducida, como en este caso, y si la cantidad de memoria RAM disponible es suficiente.

Usando los datos de la **2.3.3.1** he decidido analizar 3 valores adicionales: la media, la mediana y la desviación estándar.

Tabla 2.3.3.2. Media, mediana y desviación estándar de los tiempos de consulta

Medida	mongoDB	.json	MySQL
Media	518,4	82,5	521,8
Mediana	517,5	80	519
Desviación std.	7,8	7,9	10,4

En la **Tabla 2.3.3.2** se puede apreciar que la media y la mediana no difieren, en gran medida, una de la otra. Este resultado nos indica, a parte del valor promedio de los tiempos de consulta, que los datos son bastante simétricos y no hay valores poco comunes o atípicos. Sobre la desviación estándar, vemos que los resultados apuntan a que no hay una gran variación de los tiempos con respecto a la media. Por lo tanto, el tiempo en realizar las consultas, es bastante similar en los casos de mongoDB y del fichero .json, y algo más variado cuando se utiliza MySQL.

Pero viendo los tiempos de la **Tabla 2.3.3.1** no quedaría convencido de que implementar mongoDB es la mejor opción ya que los tiempos son 7 veces superiores respecto al fichero .json. Es por ello que realicé pruebas llevando la memoria RAM al extremo para observar cómo se comportan los 3 tipos de base de datos cuando casi no tienen memoria disponible. Con esto se podrá comprobar el comportamiento del fichero json cuando los datos ya no caben en memoria.

Para realizar las pruebas hice uso de una aplicación llamada *MemTest64* [28] que me permitía llevar al límite mi memoria RAM para poder hacer las consultas.

La siguiente **Tabla 2.3.3.3** muestra los resultados obtenidos.

Tabla 2.3.3.3 Rendimiento de las bases de datos con sobrecarga de RAM

Consulta	mongoDB	json	MySQL
Login	615	3587	625
Grupos	604	4522	659
Puntos	648	5453	663
Colección	636	6370	672
Liga	638	4525	655
FormulaUno	644	3828	698
Avatar	622	5760	668
Cuestionario	630	5384	656
Añadir	741	4893	732

alumno			
Borrar alumno	542	5134	675

En la **Tabla 2.3.3.3** se puede observar que los tiempos han cambiado mucho para el caso de json. Llegando a no ser buena opción una vez que los datos no te caben en memoria. Por el contrario, tanto mongoDB y MySQL se han mantenido en un rango muy similar al de las pruebas de la **Tabla 2.3.3.1**.

Cómo última prueba, he realizado varias consultas para comprobar el tiempo que tarda en sucesivas consultas iguales. Como se puede intuir, al tener datos en el caché, los tiempos se reducirán. La **Tabla 2.3.3.4** muestra los tiempos obtenidos a la vez que se sobrecargaba la memoria RAM.

Tabla 2.3.3.4 Rendimiento bases de datos tras sucesivas consultas y sobrecarga de RAM

Consulta	mongoDB	json	MySQL
Login	24	20	26
Grupos	24	18	25
Puntos	23	19	25
Colección	22	21	26
Liga	24	18	24
FormulaUno	24	20	27
Avatar	23	20	25
Cuestionario	23	19	26
Añadir alumno	25	22	27
Borrar alumno	30	26	30

Tal y como se pueden observar en la **Tabla 2.3.3.4**, tras realizar 2 o más consultas iguales o muy parecidas (solo modificando algún parámetro), se obtienen tiempos de espera muy bajos comparados con los valores anteriores. En este caso no hay una diferencia significativa entre bases de datos.

Tras haber analizado los diferentes tiempos de respuesta de cada prueba, he escogido a mongoDB como base de datos para guardar la información de Classpip. No solo ofrece mejores tiempos de respuesta, sino que es una base de datos muy escalable y flexible. Gracias a esto facilitamos la gestión de los datos de Classpip y la integración de nuevos tipos de datos.

2.4. Implementación de la base de datos escogida

En esta última sección explicaré la implementación realizada para poder usar mongoDB como base de datos para Classpip.

En cuanto a las tecnologías utilizadas en esta sección, son las siguientes:

- **Loopback-connector-mongodb:** para realizar la conexión entre el proyecto y la base de datos.

- **mongoDB:** para poder gestionar la base de datos.

Para la creación de la base de datos y de los modelos que se utilizan, he creado dos tutoriales que se pueden encontrar en los **Anexos 6.2 y 6.3**, uno para mongoDB y otro para MySQL. Estos manuales también explican como importar y exportar los dos tipos de base de datos.

2.4.1. Modificaciones en el servidor

Con tal de implementar y mantener las mismas funcionalidades, se tuvieron que hacer modificaciones mínimas.

MongoDB funciona con ficheros BSON [29]. De forma automática y para evitar problemas, cada fichero BSON tiene su propio id, el cual mongoDB crea automáticamente. Esto comportaba el hecho de, o bien tener que modificar los datos existentes, o bien crearlos de nuevo para que el id se asigne automáticamente.

Con estas modificaciones la relación entre los proyectos queda como muestro en la **Fig 2.4.1.1**.

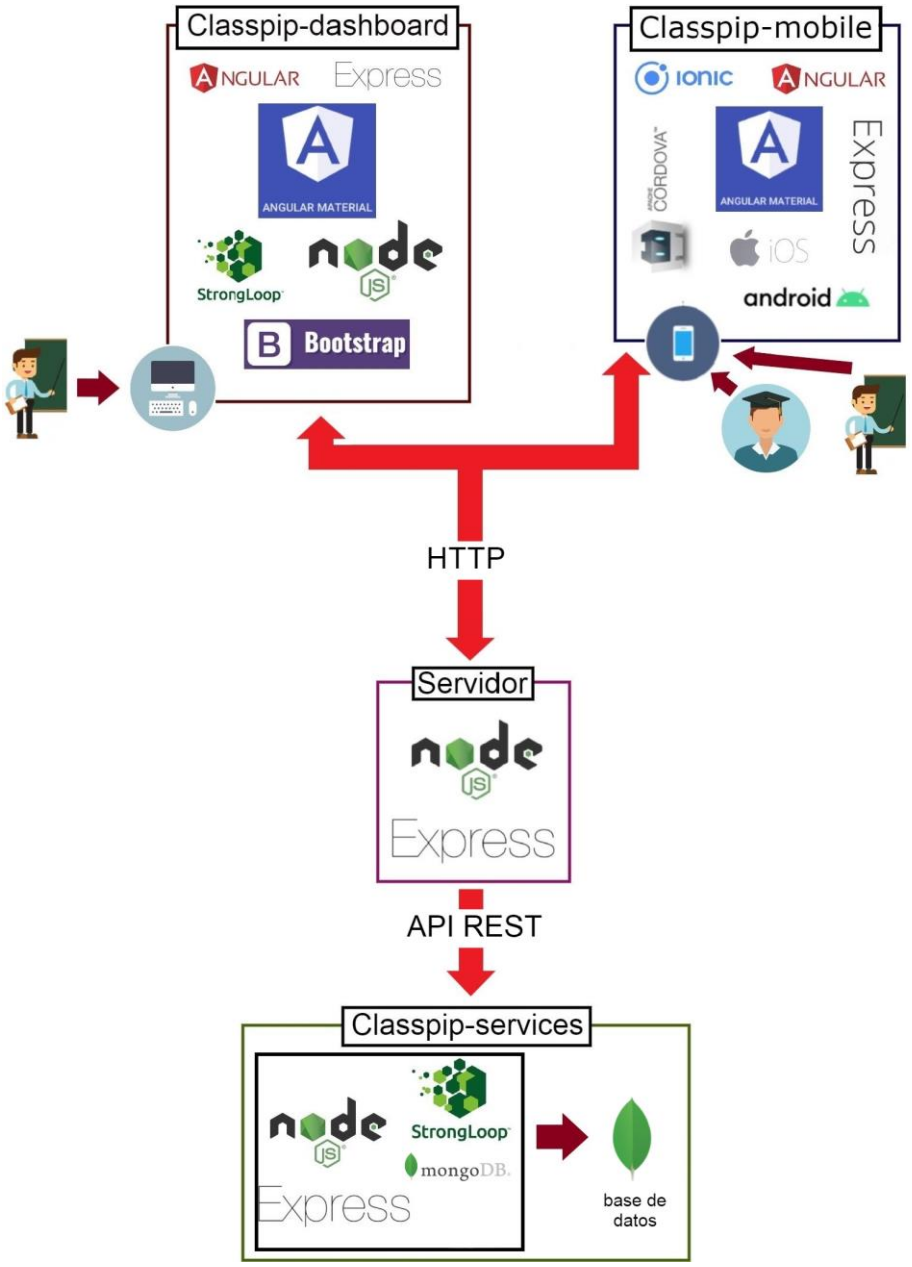

Fig 2.4.1.1 Relación entre los proyectos de Classpip con mongodb

Tal y como se puede apreciar en la **Fig 2.4.1.1**, tanto el profesor como el alumno acceden mediante el explorador web y hacen consultas HTTP al Servidor. Este, a su vez, realiza las consultas a Classpip-services para obtener finalmente los datos solicitados por el usuario a través de la nueva base de datos (mongodb).

Como producto final de esta parte del proyecto he creado dos guías paso a paso para realizar la implementación, tanto de mongoDB como de MySQL, para usarlas como base de datos para una aplicación. La guía se divide en dos secciones: una para mongodb y la otra para MySQL. En las dos guías explico cómo realizar la instalación de la base de datos, su configuración y como crear modelos para verificar el correcto funcionamiento. Al final de cada guía hay

unos apartados que enseñan como pasar la base de datos a un json y viceversa. Estas dos guías resultan de gran utilidad para poder ver cómo hacer uso de una base de datos de manera sencilla y guiada. Las dos guías, una para MySQL y otra para mongoDB, se estructuran de la siguiente manera:

- Se empieza enseñando de dónde y cómo instalar los diferentes componentes necesarios.
- Se enseña cómo crear la base de datos.
- Se crea un nuevo proyecto haciendo uso de las librerías instaladas previamente.
- Se configura el proyecto para que use la base de datos escogida.
- Se crean los modelos de datos para hacer pruebas.
- Se añaden las propiedades y las relaciones necesarias al modelo de datos.
- Se muestra cómo configurar el proyecto para que guarde los modelos en la base de datos creada.
- Se hacen pruebas, insertando datos, para demostrar su correcto funcionamiento.
- El último apartado de la guía se centra en explicar cómo exportar e importar datos desde y hacia la base de datos. También se incluye cómo exportar e importar una base de datos en su totalidad.

Concluyo este apartado con una reflexión sobre esta sección. La implementación fue bastante fácil de realizar ya que no hubo grandes problemas. En cuanto a escoger la base de datos más indicada, sí que tuve que pensar tanto en aspectos actuales como en futuros. El cambio realmente no se nota hasta el momento de gestionar los datos o querer hacer alguna modificación manual. Es entonces cuando vemos la gran facilidad que comporta mongoDB, así como su sencillez.

CAPÍTULO 3. LTI

Un alumno accede a su plataforma de aprendizaje favorita, por ejemplo, Moodle, para consultar sus tareas pendientes o para ver las notas que ha obtenido en sus exámenes recientes. Tras realizar la consulta y quedar satisfecho con su rendimiento, le entran ganas de poder ver su avatar. Pero este avatar lo tiene dentro de una aplicación del móvil llamada Classpip. Entonces el alumno tiene que encender el teléfono, iniciar la aplicación, hacer el inicio de sesión con sus credenciales y finalmente ir al apartado dónde tiene su Avatar. Pero todo esto se puede simplificar de manera muy práctica si hacemos uso de un estándar de comunicación llamado LTI.

LTI nos ofrece una vía de comunicar herramientas (aplicación Classpip) con sistemas de aprendizaje e-learning (Moodle) sin tener que salir de este último.

Esto resulta muy útil ya que el alumno podrá acceder a ver su Avatar sin tener que salir de Moodle, consiguiendo una reducción importante del tiempo gastado y mejorando su satisfacción con la plataforma.

Es entonces cuando surge la necesidad de implementar este protocolo para conectar una plataforma de aprendizaje con una aplicación.

3.1. LTI

La Interoperabilidad entre Herramientas de Aprendizaje o LTI (Learning Tools Interoperability), es un estándar de comunicación entre herramientas y plataformas de aprendizaje electrónico. Fue desarrollado por una organización de carácter mundial, que establece los diferentes estándares de desarrollo de tecnologías educativas, conocido con el nombre de IMS Global Consorsium [30].

Este estándar nos permite integrar aplicaciones de aprendizaje o herramientas (tools), las cuales son proporcionadas por proveedores de herramientas (tool providers), en nuestro caso Classpip, para ser utilizadas en plataformas de aprendizaje llamadas consumidores de herramienta (tool consumers), en nuestro caso Moodle.

A la hora de escoger qué librería era la mejor para poder realizar esta implementación, me di cuenta de que había una que cumplía con los estándares básicos. Esta librería se llama LTIJS y la ventaja respecto a la gran mayoría es que es muy sencilla de utilizar y además usa la última versión de LTI, la versión 1.3. Es una gran ventaja ya que las anteriores no solo están obsoletas, sino que algunas no permiten el uso de comunicación vía REST y tienen ciertos problemas de seguridad, los cuales no comentaré en este proyecto.

3.2. LTIJS

LTIJS es una librería que implementa un servidor Express propio (aunque su uso no es requerido) junto a rutas preconfiguradas para poder manejar el protocolo LTI 1.3. Esto permite que no nos tengamos que preocupar por las implementaciones manuales de seguridad o validación necesarias.

Esta librería nativamente usa mongoDB para guardar y manejar los datos del servidor.

Con tal de dar solución al problema planteado con anterioridad, tendré las siguientes premisas:

- Tool consumer (consumidor de la herramienta): haré uso de la plataforma de aprendizaje (LMS) usada por la UPC, Moodle. Esto me va a permitir mostrar la manera de realizar la implementación en un entorno específico. Se tiene que tener en cuenta que esta implementación es independiente de la plataforma que se escoja ya que cualquier plataforma que permita el uso de LTI, sirve para este propósito.

- Tool provider (proveedor de herramienta): utilizaré a Classpip por lo que los juegos que tenemos formarán, en conjunto, lo que podremos utilizar en Moodle.

Tras realizar la implementación, la forma en la que se comunican y estructuran los elementos implicados la podemos ver resumida en la **Fig 3.1.1**.

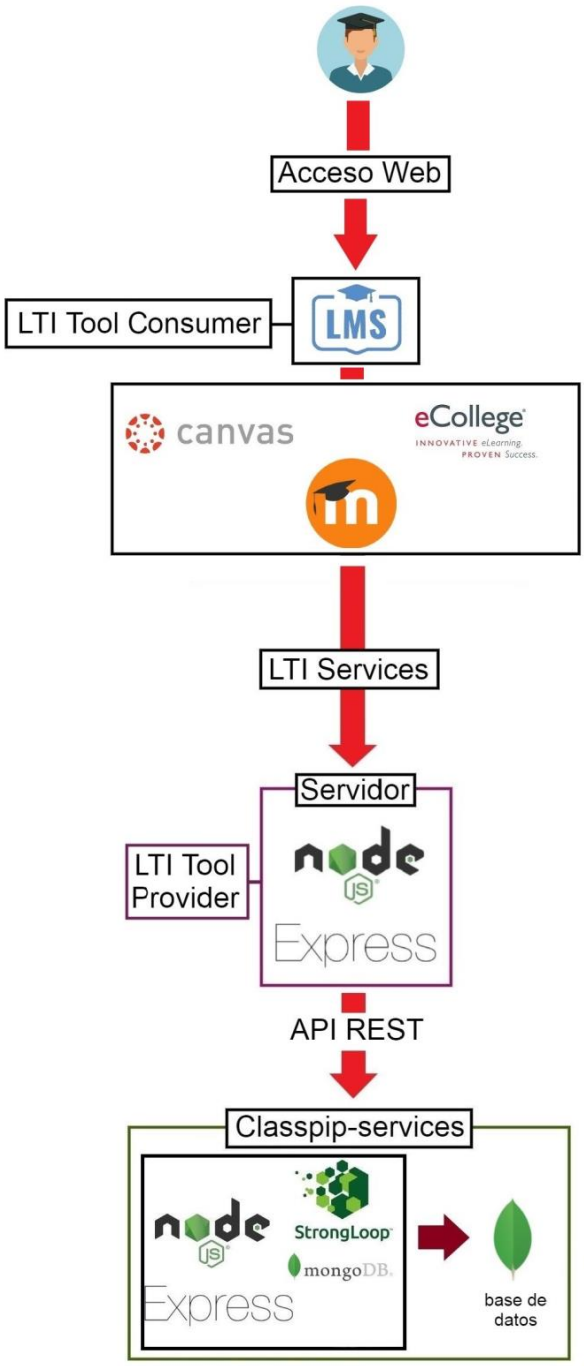

Fig 3.1.1 Comunicación entre los elementos de LTI

Como podemos observar en la **Fig 3.1.1**, el usuario comienza haciendo uso de la plataforma de aprendizaje (Moodle) a través de un explorar web. Entonces, el consumidor de LTI (Moodle) y el proveedor de la herramienta (Classpip) se comunican usando los servicios LTI implementados por LTIJS (LTI Services). Luego, el Servidor, se comunica haciendo uso de la API REST con Classpip-services.

En la siguiente sección veremos la utilidad de LTI mediante el uso de LTIJS.

3.3. Utilidad *de la herramienta LTIJS*

Así como se ha podido observar del ejemplo del alumno que quiere ver su Avatar dentro de Moodle, hacer uso de LTIJS nos abre un camino a muchas posibilidades futuras. Esto es debido a que esta implementación permite ver la forma en la que se enlazan los diferentes componentes de LTI. De esta forma conseguimos poder utilizar un número indefinido de aplicaciones dentro de nuestra plataforma de aprendizaje.

En resumidas cuentas, LTIJS es una herramienta que permitirá la entrada de cualquier aplicación deseada dentro del ámbito de las plataformas de aprendizaje, y más específicamente, en Moodle. Con el objetivo de poder implementarlo, en un futuro, dentro del Moodle de la UPC.

En el siguiente apartado explicaré las 6 fases (retos) en las que dividí la implementación, haciendo hincapié en los resultados que quería obtener.

3.4. 6 retos

Los diferentes objetivos que se plantearon en un principio fueron los siguientes:

- **Reto #1 - Login del usuario de Moodle en Classpip:** el objetivo principal es poder mapear correctamente los usuarios de Moodle con los de Classpip.

- **Reto #2 - Ver lista de juegos de un alumno:** visualizar en el navegador una lista completa de los juegos disponibles de un alumno en concreto.

- **Reto #3 - Ver ranking de un juego:** visualizar el ranking de puntos de un juego determinado.

- **Reto #4 - Ver/Contestar un juego de cuestionario:** visualizar y responder las preguntas de un determinado juego de cuestionario.

- **Reto #5 - Ver una colección de cromos:** visualizar la colección de un alumno y sus cromos disponibles (con y sin doble cara).

- **Reto #6 - Ver el avatar del alumno:** visualizar el avatar, sus complementos y la nota de voz (en caso de estar disponible).

Estos objetivos me ayudaron a centrar y a visualizar, de una manera más práctica, los requerimientos que me hacían falta para llevar realizar la implementación. Esto es debido a que cada reto implica sus particularidades para así conseguir tener un abanico de opciones en cuanto a formas de extraer y visualizar los diferentes tipos de juegos de Classpip. Siendo estos los resultados que quería obtener.

Para llevar a cabo los diferentes retos planteados, tuve que hacer uso de una librería de LTI conjuntamente con la plataforma Moodle en local. Esto significa que también explicaré cómo configuré dicha plataforma para el uso de una herramienta externa. Hablaré de todo ello en el siguiente apartado.

También he realizado una guía, más centrada en el desarrollo de cada uno de los componentes de LTI necesarios para superar los retos, la cual se encuentra en el anexo.

CAPÍTULO 4. IMPLEMENTACIÓN DE LOS RETOS

En esta sección explicaré los pasos que he seguido para conseguir implementar, dentro de Moodle, los diferentes juegos de Classpip. Todo ello unido por la herramienta LTIJS, la cual nos permite enlazar dichos contenidos entre sí.

4.1. Arquitectura de la solución

Las tecnologías de las cuales tuve que hacer uso para permitir la interacción entre LTI y Moodle son las siguientes:

- **Moodle:** uno de los muchos tipos de plataforma de aprendizaje. Necesaria para poder hacer la implementación.
- **LTIJS:** librería para implementar la herramienta LTI (v1.3) en nuestro proyecto.
- **Dotenv:** para poder cargar variables de entorno desde un fichero *.env* (requerido para facilitar el uso de LTIJS).
- **Babel-eslint:** para soportar las características que no tenemos en ESLint y poder parsear correctamente el valor de =>.
- **Mongodb:** base de datos soportada por defecto en LTI.
- **EJS:** Motor de modelos para generar páginas HTML y tener acceso a variables del servidor.

Estos cambios fueron realizados en la parte del servidor, el cual crea y pone en marcha el servidor de la aplicación de Classpip. Es la forma más coherente de implementar esta herramienta ya que es el servidor el que permitirá la conectividad LTI para poder acceder al proyecto de Services (en donde encontramos la API REST para poder actuar sobre los datos).

En la **Fig 4.1.1** podemos ver el flujo que se establece.

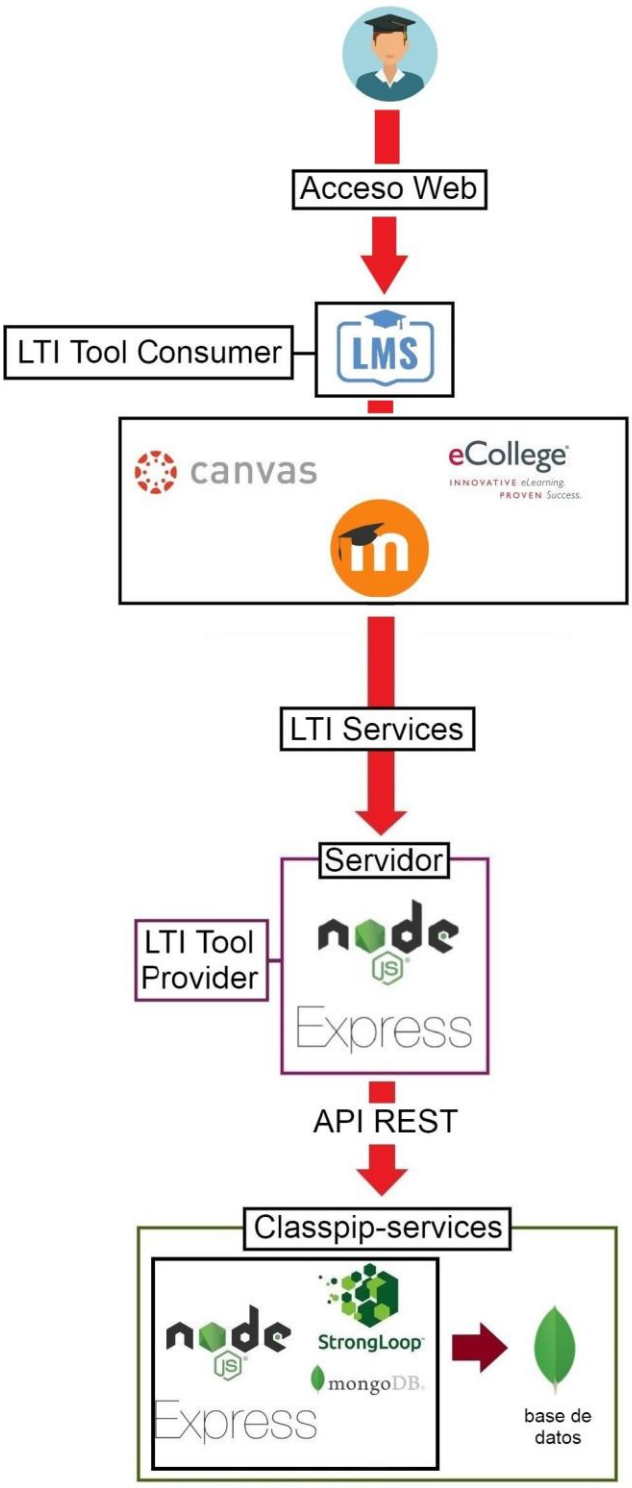

Fig 4.1.1 Flujo entre el servidor, Moodle y el proyecto Classpip

Tal y como se observa en la **Fig 4.1.1**, el servidor es el que implementa el protocolo LTI conjuntamente a Moodle. Aunque sea el servidor el que realice la conexión LTI, los datos se extraen mediante el servicio API REST del proyecto Classpip Services.

Ahora me centraré en explicar, en 3 secciones separadas, de qué forma he realizado la implementación de la librería LTIJS. El objetivo que me propuse para el apartado de LTI lo dividiré en las siguientes partes:

- **Cuestiones comunes:** donde explicaré aquello que es común para poder implementar los retos. Aquí entra toda la instalación y puesta en marcha, tanto de Moodle como de LTIJS.

- **Retos:** estarán los objetivos de los 6 retos y su solución, así como las pruebas realizadas.

4.2. Cuestiones comunes

En este apartado trataré las partes comunes que se comparten en los retos planteados.

Con tal de poder realizar la implementación lo primero y más básico es realizar la instalación de los componentes mencionados en el apartado **4.1**. Para este apartado son necesarios los siguientes componentes:

- **Moodle**

- **LTIJS y dotenv**

Tras instalarlos se configuran por partes, primero se empieza con Moodle para poder extraer los datos necesarios que necesita la función del Servidor que conecta Classpip con Moodle. Luego se configura la página a dónde se redirecciona al usuario una vez se establece la conexión entre Classpip y Moodle. Dentro de la ruta configurada, se establece la página o el contenido que se quiere mostrar. De este apartado se obtiene la clave pública que tendremos que introducir dentro de la configuración de Moodle.

Tras seguir todos los pasos, se crea una nueva actividad dentro de Moodle y se establece de forma que haga uso de la nueva herramienta externa (Classpip).

Una guía más detallada para poder instalar y poner en marcha tanto Moodle como LTIJS, se encuentra en el **Anexo 4**. En este apartado, se ven los pasos a seguir para poder implementar y conectar Moodle con Classpip.

4.2.1. Pruebas

Lo primero es comprobar que tenemos acceso a Moodle en local:

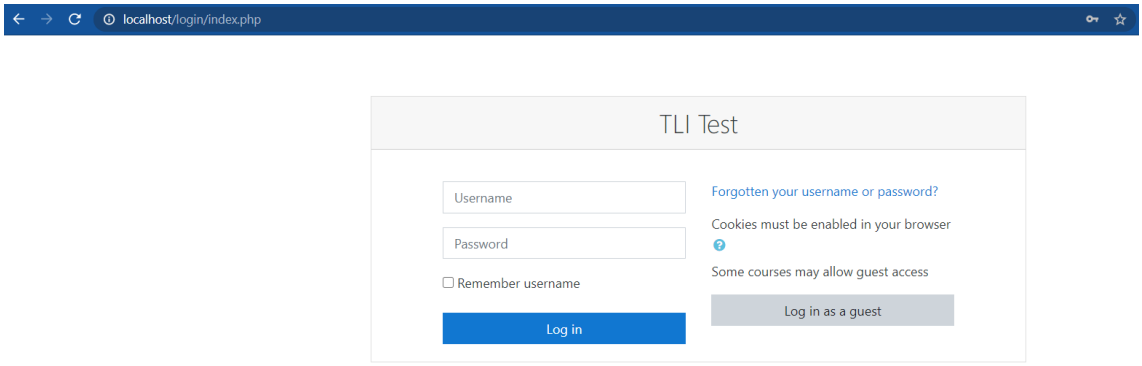

Fig 4.2.1.1 Login de Moodle

Tal y como podemos ver en la Fig 4.2.1.1, Moodle está iniciado en mi entorno local (localhost).

La siguiente prueba consiste en ver si Classpip y Moodle están conectados. Para ello, tal como se puede observar en la guía del **Anexo 5**, se tiene que configurar la herramienta externa (Classpip) en Moodle y crear una actividad que use dicha herramienta.

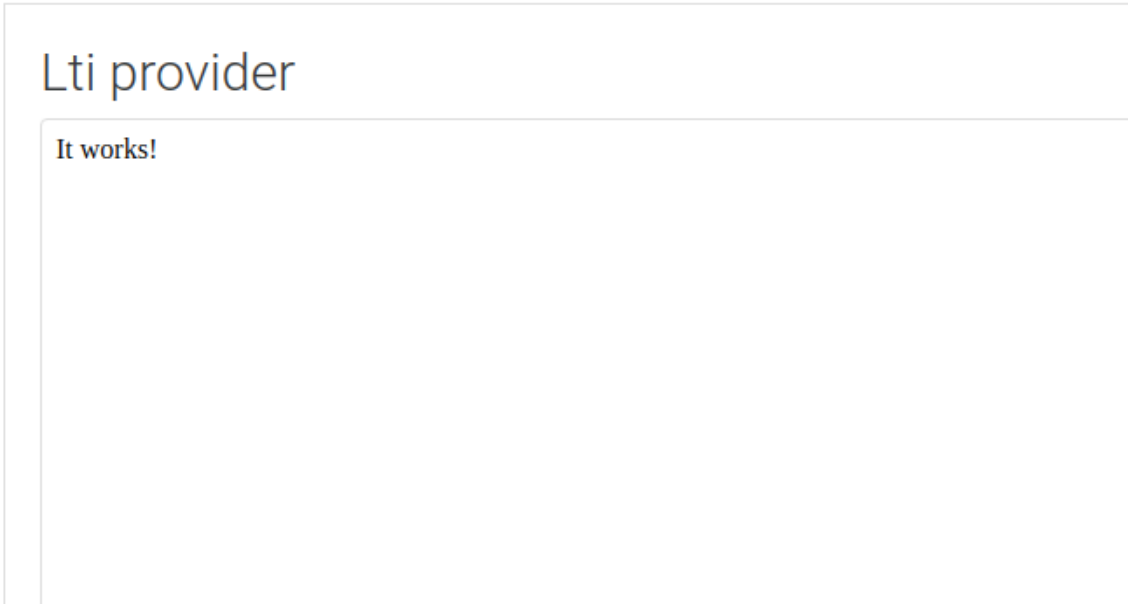

Fig 4.2.1.2 Actividad LTI dentro de Moodle

Podemos ver como en la **Fig 4.2.1.2** recibimos un mensaje directamente del Servidor de Classpip.

Con estas dos pruebas funcionando, continuaré explicando los diferentes retos planteados.

4.3. Reto #1: Inicio de sesión

En este apartado explicaré como se realiza el inició de sesión para poder mapear correctamente el usuario de Moodle con el usuario de Classpip (dentro de la base de datos de mongoDB). Para ello necesitaremos los datos del usuario de Moodle y consultar datos en la base de datos de Classpip, así como modificar el usuario acorde a si ha sido registrado.

4.3.1. Configuración de la página de Login

Para poder instalar los componentes necesarios y hacer la configuración requerida, he creado una guía que se encuentra en el **Anexo 6**.

En este primer reto, el mapeo de usuarios finalmente se hará cogiendo los datos del usuario que inicie sesión en Moodle para luego comprobar, en la base de datos de Classpip, si ya ha sido registrado en Moodle. Esto lo hará una función que comprueba si tiene o no un id asignado en el campo *UsuarioMoodle*. En caso de no estarlo, le ofrece al alumno un usuario y contraseña que extrae de sus datos dentro de Moodle. Por otro lado, si ya está registrado, cargaremos la primera pantalla de todas, en donde el alumno podrá ver todos los juegos que tiene disponibles. Antes de cargar la siguiente página, el Servidor guardará los nuevos datos del Alumno. En este caso obtendrá el id único del usuario de Moodle y lo guardará dentro del campo *UsuarioMoodle* del alumno que haya iniciado sesión.

4.3.2. Pruebas

Nada más iniciar la actividad, veremos una ventana emergente con cierta información, tal y como se puede observar en la **Fig 4.2.3.2.1**.

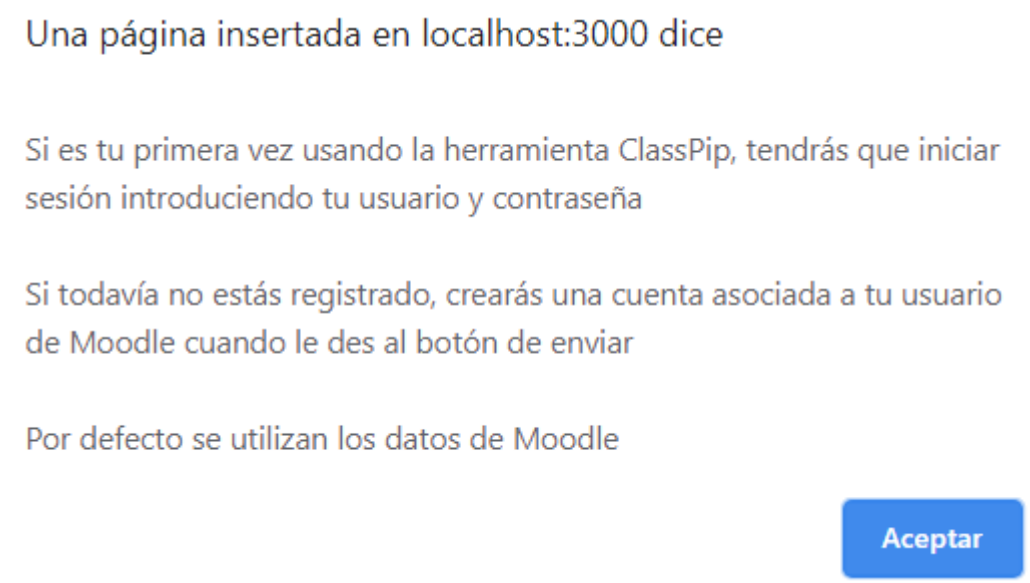

Fig 4.3.2.1 Ventana emergente de inicio de sesión

El formulario de inicio de sesión, por defecto, coge los datos del usuario de Moodle (**Fig 4.3.2.2**).

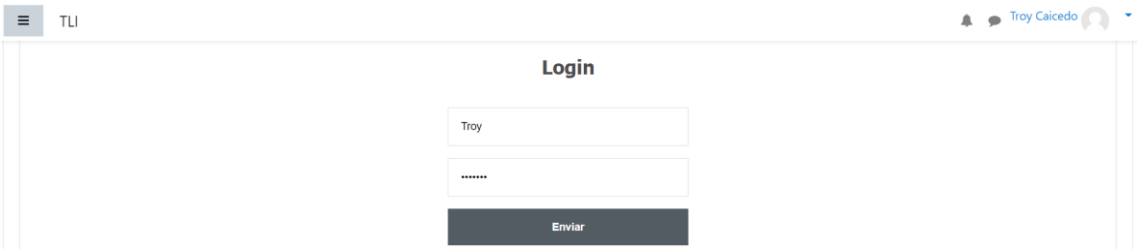

Fig 4.3.2.2 Inicio de sesión

En la base de datos de Classpip **Fig 4.3.2.3**, mongoDB, podemos ver que el campo de *UsuarioMoodle* contiene un id.


```
_id: ObjectId("5f060b36df47712d3c799a4b")
Nombre: "Fidu"
PrimerApellido: "Caicedo"
SegundoApellido: "Uvidia"
ImagenPerfil: "Imperialdramon.png"
UsuarioMoodle: ObjectId("5f5592a2f0f4a763b8786cdf")
profesorId: ObjectId("5f0ddbb9aa85250b704e6b95")
```


Fig 4.3.2.3 Datos actualizados en mongoDB

Verificando el correcto funcionamiento del inicio de sesión (**Fig 4.3.2.2**) y guardado de datos en mongoDB (**Fig 4.3.2.3**), podemos pasar a ver el segundo de los 6 retos.

4.4. Reto #2: Ver lista de juegos

Este segundo reto consiste en mostrar la lista de juegos del alumno que haya iniciado sesión. Para ello necesitamos obtener los datos de la base de datos y posteriormente mostrarlos en la ventana de Moodle.

4.4.1. Configuración página de lista de juegos

Para poder instalar los componentes necesarios y hacer la configuración requerida he creado una guía que se encuentra en el **Anexo 7**.

En este reto, se ha creado una función para recoger todos los datos de los juegos de un determinado alumno y luego se ha implementado la página *ejs* para poder mostrarla en Moodle. Contiene varias funciones que sirven para recoger los datos necesarios para cada juego. En la guía se explican las funciones del Servidor y luego las funciones de la página ejs.

4.4.2. Pruebas

Una vez iniciada sesión, se nos muestra una bienvenida con el nombre del alumno y su foto de perfil, así como 3 listas con los juegos disponibles (**Fig 4.4.2.1**). En caso de no tener una foto de perfil asignada, se mostrará una por defecto.

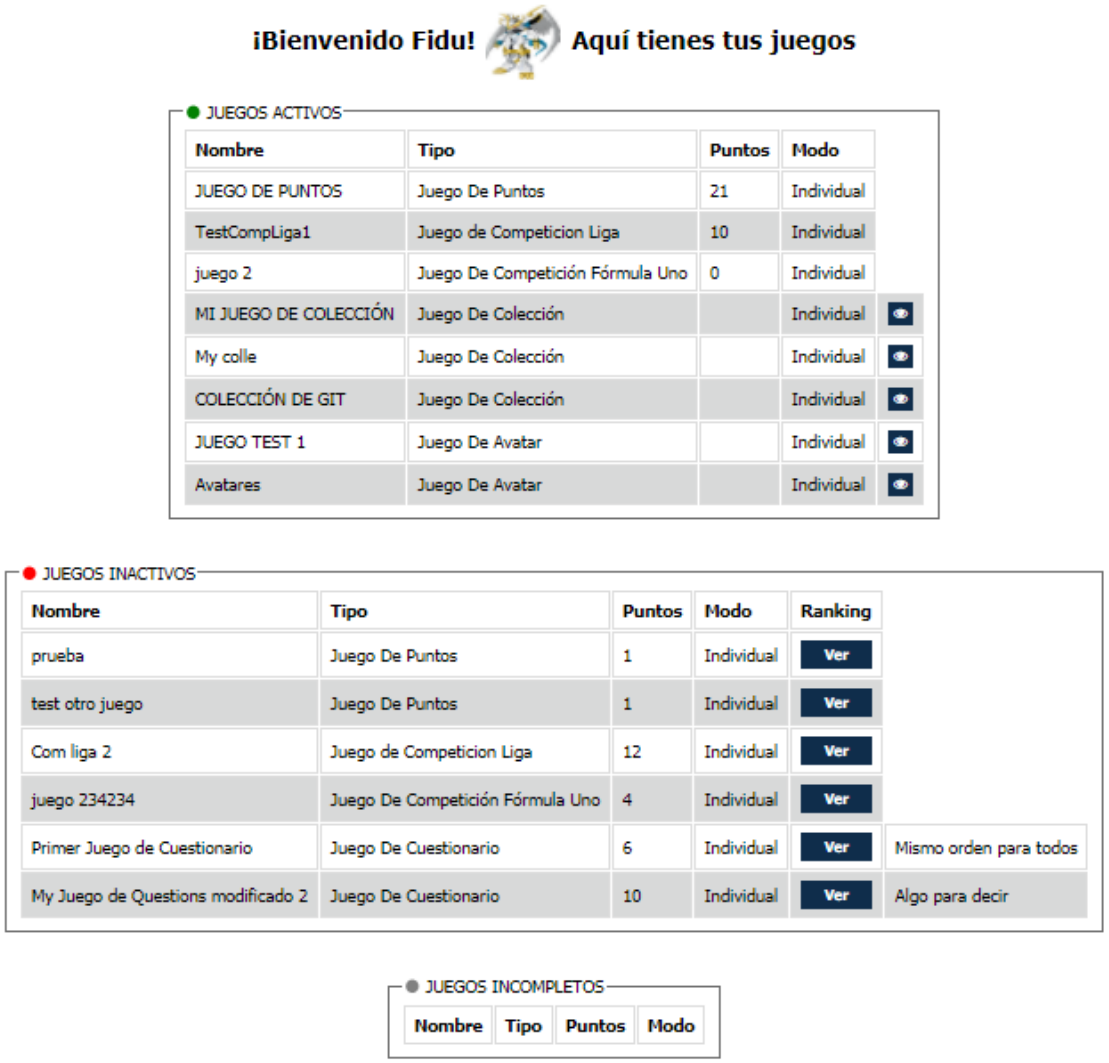

Fig 4.4.2.1 Lista de juegos

Con esto, se considera completado el segundo reto por lo que se puede continuar con el siguiente.

4.5. *Reto #3: Ver ranking de un juego*

El siguiente reto es el de visualizar el ranking de un juego. Para ello también se consultará a la base de datos de Classpip y se mostrará el contenido en la ventana de Moodle.

4.5.1. Configuración página de lista de juegos

Para poder instalar los componentes necesarios y hacer la configuración requerida he creado una guía que se encuentra en el **Anexo 8**.

Igual que con el anterior reto, lo primero es crear la ruta y luego la página ejs. Para ello se crean diversas funciones que extraen los datos necesarios y luego se procesa de forma que la lista quede ordenada por puntos. Una vez se tiene la lista ordenada se pasa como variable a la página ejs para que tenga acceso a los datos. A partir de aquí se configura la página para que lo muestre dentro de una tabla que contenga el nombre del jugador y su puntuación.

4.5.2. Pruebas

Nada más cargar la página se le indica al alumno lo que verá a continuación Fig 4.5.2.1.

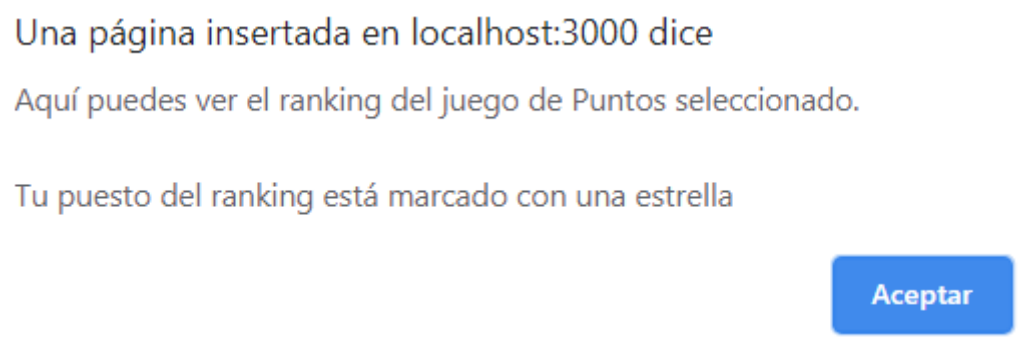

Fig 4.5.2.1 Mensaje informativo para el alumno

El ranking de puntos queda tal y como se muestra en la **Fig 4.5.2.2.**

Fig 4.5.2.2 Ranking de puntos

He añadido el detalle de colocar una estrella negra junto al nombre del alumno que ha seleccionado ver el ranking. También se marca el top 3 mediante colores: oro, plata y bronce.

Con esto también se da por superado el reto #3.

4.6. Reto #4: Ver/Contestar juego de cuestionario

En esta sección me centraré en uno de los retos que implicó más atención al detalle: los cuestionarios. En comparación al resto de retos (a excepción del primero), este implica actualizar la información de un usuario determinado para poder guardar la puntuación y las respuestas correctas e incorrectas del alumno.

4.6.1. Configuración página de lista de juegos

Como en los anteriores retos, se ha creado una función para extraer la información de la base de datos y guardarla, para poder pasarla a la página *ejs* que mostrará la información en Moodle. Toda la implementación y configuración necesaria se encuentra en el **Anexo 9**.

Hay que destacar que es necesario guardar las respuestas y, por tanto, modificar los datos de las colecciones relacionadas con el cuestionario. En concreto las colecciones que guardan los resultados o la puntuación obtenida. Para ello se crea una función que coge los datos del juego de cuestionario del alumno y crea un objeto en formato JSON que contiene los datos anteriores y las respuestas o los puntos del alumno, dependiendo de qué colección se quiere actualizar.

4.6.2. Pruebas

Primero de todo tenemos que acceder al cuestionario desde la página con la lista de juegos tal y como se muestra en la **Fig 4.6.2.1**.

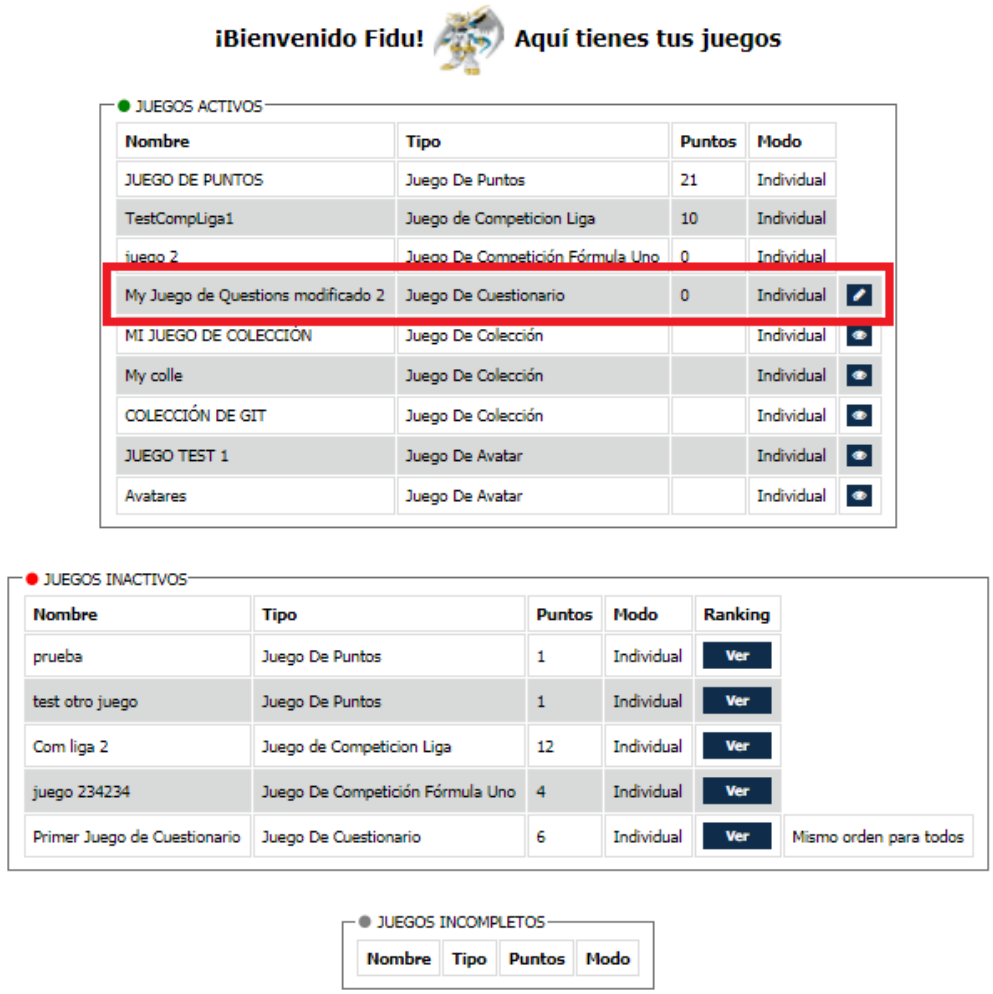

Fig 4.6.2.1 Listado de juegos

Se puede ver el resultado final en la **Fig 4.6.2.2**.

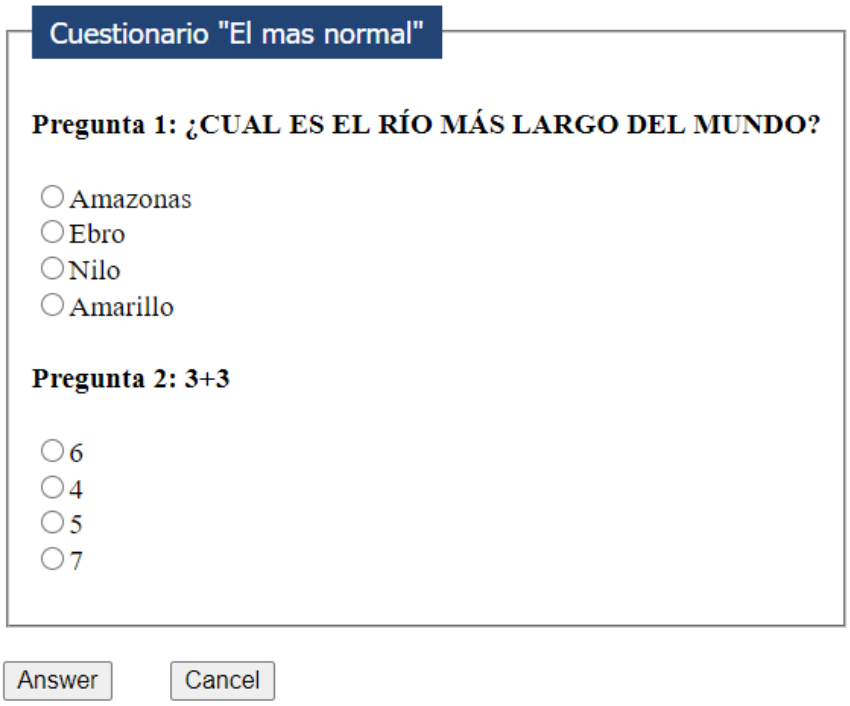

Fig 4.6.2.2 Ranking de puntos

He decidido juntar todas las preguntas que pertenezcan al mismo cuestionario en un solo apartado.

En caso de no seleccionar una respuesta por pregunta, sale una alerta tal y como se puede observar en la **Fig 4.6.2.3**.

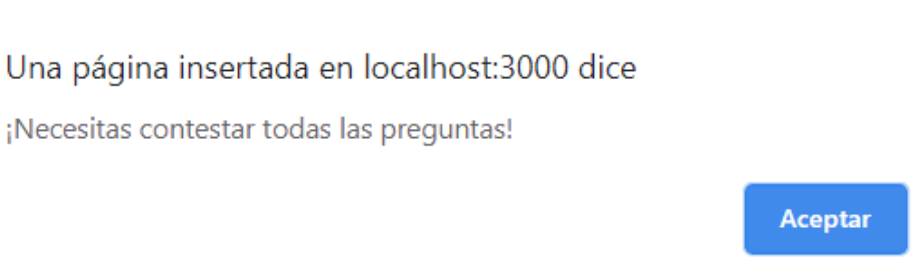

Fig 4.6.2.3 Alerta avisando que faltan preguntas pendientes

Los datos también se actualizan en la base de datos y, tras responder el cuestionario, se redirige al usuario de nuevo a la lista de juegos actualizada (**Fig 4.6.2.4**).

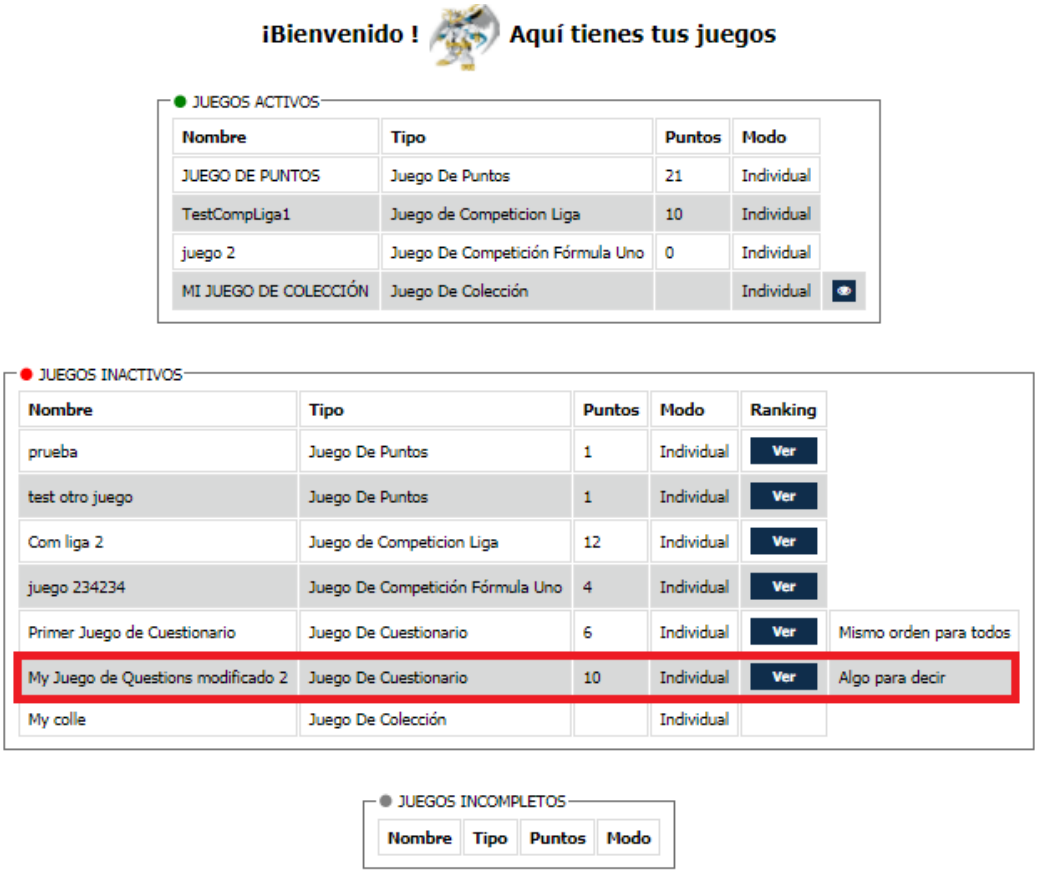

Fig 4.6.2.4 Página actualizada tras responder el cuestionario

En la **Fig 4.6.2.4** se marca en rojo la nueva puntuación del alumno y podemos ver que el juego aparece en la tabla de juegos inactivos.

Con esto también se da por superado el reto #4.

4.7. Reto #5: Ver juego de colección

El objetivo de este reto es poder ver una colección de cromos de un alumno. Para ello tendremos que consultar la base de datos para ver qué cromos componen la colección y qué cromos tiene el alumno. Tras esto, los mostraremos en la ventana de Moodle mediante páginas ejs.

4.8.1. Configuración página de Colección

Como en los anteriores retos, toda la implementación y configuración necesaria se encuentra en el **Anexo 10**. En este caso fue necesario crear lo mismo que en los anteriores casos: una función para extraer los datos de la base de datos y una configuración correcta de las páginas ejs. Hay que destacar que en este caso se trata con imágenes por lo que se tienen que colocar de forma ordenada como si fuera un álbum físico. También se tienen que mostrar, en

escala de grises, aquellos cromos que el alumno no haya obtenido aún y si se pasa el ratón por encima, se debe ver el reverso del cromo en caso de existir.

4.8.2. Pruebas

Como se puede ver en la **Fig 4.8.2.1**, el resultado es una colección de cromos en la que el cromo que el alumno tiene queda resaltado en color.

Fig 4.8.2.1 Colección de cromos (vista inicial)

Tras esto, la siguiente parte del reto es conseguir que al pasar por encima se muestre el reverso del cromo. Se puede apreciar en la **Fig 4.8.2.2**.

Fig 4.8.2.2 Colección de cromos (vista trasera del cromo seleccionado)

Con estas dos partes del reto resueltas, el reto queda completado.

4.8. Reto #6: Ver juego de avatar

El reto final consiste en ser capaz de visualizar el avatar de un alumno en concreto. Tenemos que tener en cuenta de que tratamos con diferentes imágenes que tienes que estar superpuestas de manera correcta. Un pequeño detalle es que el alumno no siempre tiene los privilegios para ver todo el contenido de su avatar.

4.9.1. Configuración página de Avatar

Esta última implementación y configuración se encuentra en el **Anexo 11**. En este caso fue necesario crear tanto una función para extraer los datos, como una configuración para poder visualizarlos.

Tal y como se pudo ver en el anterior reto de ver el juego de Colección, este reto también implica manejar imágenes. Las dos peculiaridades son:

- Todas las imágenes se tienen que superponer de manera correcta.
- En caso de tenerlo disponible, se mostrará un reproductor para el fichero de voz.

4.9.2. Pruebas

Finalmente, tras las implementaciones que se pueden ver en al apartado anterior, el resultado es el que muestro en la **Fig 4.9.2.1**.

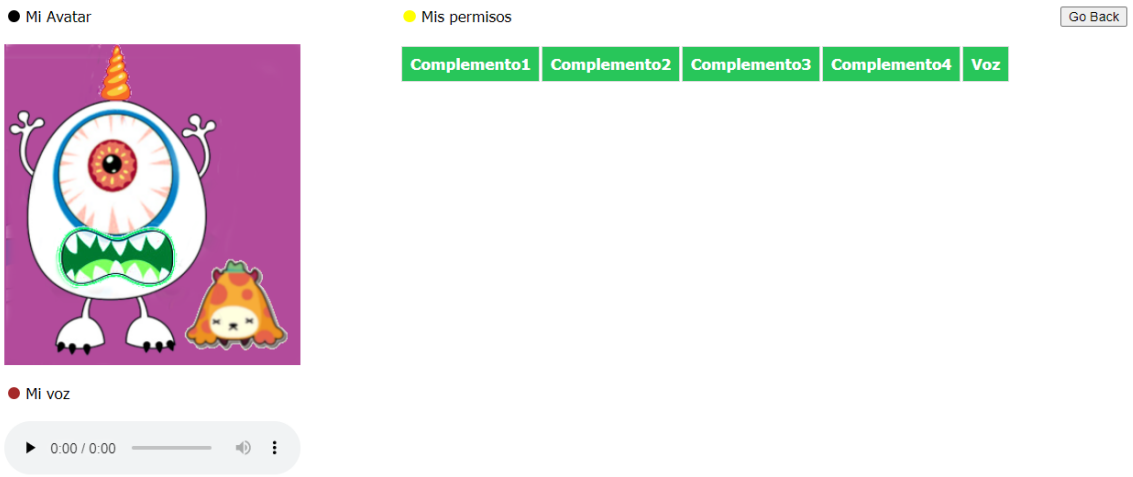

Fig 4.9.2.1 Juego de Avatar

En la **Fig 4.9.2.1** se puede ver que también incluí una lista con los permisos actuales del alumno.

En el caso de que el alumno no haya conseguido un privilegio concreto, el resultado cambia un poco, tal y como se puede ver en la **Fig 4.9.2.2**.

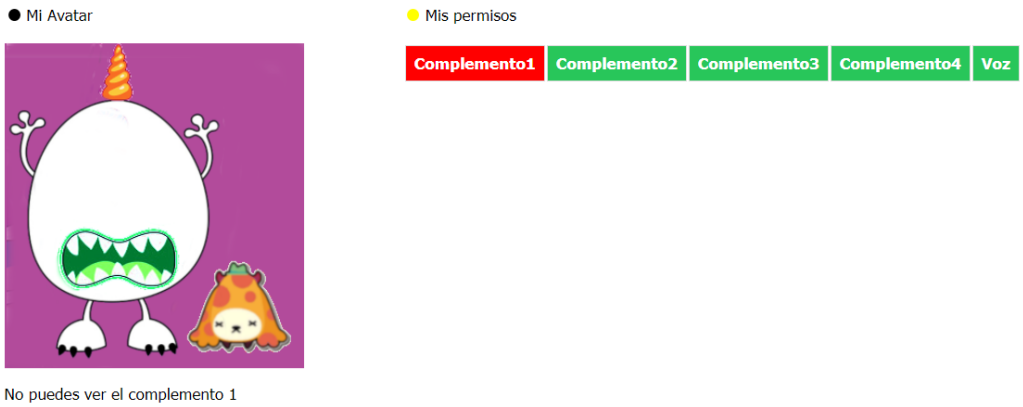

Fig 4.9.2.2 Juego de Avatar (privilegio no conseguido)

Con esto realizado, se consideran completados todos los retos planteados.

Quiero puntualizar lo siguiente: pese a que en este ejemplo lo que se consigue es visualizar o no un cierto elemento, o conjunto de ellos, la intención es la de poder habilitar o no una función concreta. En este caso sería la de poder visualizar partes del juego de Avatar, pero en Classpip la idea central es la de poder o no modificar un componente según los permisos que tenga el alumno. Ya que no era el objetivo, decidí hacerlo de esta manera para dar un ejemplo de cómo deshabilitar cierto contenido.

Tras finalizar esta implementación, puedo ver que LTIJS facilita mucho la integración de LTI y que ofrece una gran variedad de posibilidades. Cada uno de los retos ha supuesto realizar diferentes tipos de ajustes y modificaciones, ya que ni la información ni la forma de presentarla eran las mismas.

CAPÍTULO 5. CONCLUSIONES

Dividiré las conclusiones en 3 apartados: el primero a nivel más técnico, el segundo a nivel más personal y el último serán posibles futuras mejoras. Con esto podré reflejar de manera más precisa los resultados obtenidos, así como los objetivos que se habían planteado al principio.

5.1. Conclusiones técnicas

En este proyecto tenía dos objetivos diferenciados: por una parte, implementar una base de datos para Classpip y, por otra, implementar una herramienta que me permita la conexión LTI entre una aplicación y una plataforma de aprendizaje.

Partiendo de los resultados obtenidos y analizados en el capítulo 2, se puede decir que la nueva base de datos implica una gran mejora respecto al uso de un fichero .json en local. Esto permitirá más control y flexibilidad sobre los datos de Classpip. También se pudo observar cómo los tiempos de respuesta se reducen, al menos en el caso en el que el fichero json no quepa en memoria RAM. Como productos finales tenemos:

- Guía para implementar mongoDB o MySQL como base de datos para una aplicación.

Sobre la implementación de LTI puedo decir que los objetivos se han conseguido por completo. Tal y como expliqué en el capítulo 4, los diferentes retos se pudieron cumplir. Esto ofrece una base desde la cual empezar a desarrollar una implementación más ligada a Classpip y con otras funcionalidades que no se han explorado en este proyecto. Como productos finales tenemos:

- Una guía para poner en marcha Moodle y LTIJS.

- Diversas guías para poder implementar cada uno de los retos planteados. Esto ayudará a poder implementar lo necesario de una manera eficaz y directa.

Debido a la infinidad de opciones de configuración que se podían implementar, hay muchas mejoras posibles con respecto a lo que pude conseguir. Empezando por mejorar el aspecto visual de las diferentes páginas o añadiendo funciones que no se llegaron a implementar.

Todo ello queda como propuestas para futuras implementaciones.

5.2. Conclusiones personales

Cuando empecé este proyecto lo hice con muchas ganas sabiendo que esto ayudaría no solo al proyecto de Classpip, sino que a muchos otros que puedan venir.

Ya desde un principio me pareció muy interesante la idea de investigar soluciones a diferentes problemas o necesidades que se planteaban. No solo porque un ingeniero debe ser capaz de encontrar la solución más adecuada para un problema, sino también porque me ha permitido entrar en contacto con tecnologías que de otra forma sería difícil que pasara. Un ejemplo sería la librería LTIJS, la cual es bastante específica para plataformas de aprendizaje.

Durante la realización de este proyecto también intenté centrarme en programar de una manera eficaz y teniendo en mente que esto puede ser reutilizado por otras personas. Es por ello que añadí las modificaciones necesarias para que los diferentes apartados sean lo más entendibles posible.

En general el proyecto lo he podido ir desarrollando sin muchos contratiempos. A pesar del tiempo ajustado del que disponía y de las muchas distracciones que podían surgir al o largo del día, estoy muy contento de haber podido finalizar este trabajo. Ha sido un proyecto que me ha ayudado a optimizar aún más mi tiempo y que ha requerido de gran esfuerzo para finalizar.

Las dificultades más grandes que me he encontrado han sido a la hora de completar algunos de los retos, en concreto el del cuestionario y el de las imágenes.

Estas implementaciones espero que sirvan de guía para que, en un futuro, se puedan realizar mejores o nuevas funciones que ayuden a enriquecer la aplicación de Classpip.

5.3. Futuras mejoras

Posibles mejoras que podrían implementarse en este proyecto serían las siguientes:

- **Mejorar la visualización de las diferentes páginas:** implementar mejores estilos y formas de presentar los datos. Cada página se podría ir ajustando para que quedase lo más parecido a lo que ven los alumnos en sus móviles.

- **Implementar el resto de funcionalidades de Classpip:** hay funciones que no he llegado a implementar, como la modificación del avatar.

- **Corrección de errores:** hay partes que son susceptibles a fallos que podrían hacer que alguna página no cargase si se le pasan datos erróneos.

- **Implementar el resto de rankings:** en este proyecto solo he llegado a implementar el ranking del Juego de Puntos.

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

ANNEXOS

TÍTULO DEL TFG: Mejora de la base de datos de una herramienta de gamificación e implementación de una conexión con una plataforma de aprendizaje

TITULACIÓN: Grado en Ingeniería de Telemática

AUTOR: Andrés Fernando Caicedo Uvidia

DIRECTOR: Miguel Valero

FECHA: 15 de septiembre del 2020

Aquí se encuentran todos los Anexos que ayudarán a conseguir implementar las diferentes tecnologías utilizadas en este proyecto. Así mismo, empezaré con una bibliografía de referencias en la que menciono conceptos clave que han surgido a lo largo de la memoria.

La lista completa de los anexos es la siguiente:

1. **Bibliografía**
2. **Guía paso a paso para el uso de mongodb como BBDD para modelos de datos**
3. **Guía paso a paso para el uso de MySQL como BBDD para modelos de datos**
4. **Guía para instalar y poner en marcha Moodle y LTIJS**
5. **Guía para implementar el Inicio de sesión**
6. **Guía para implementar la lista de juegos de un alumno**
7. **Guía para implementar el ranking de un juego**
8. **Guía para implementar el ver/contestar un juego de cuestionario**
9. **Guía para implementar la visualización de la colección de cromos**
10. **Guía para implementar la visualización del Avatar**

1. Bibliografía

[1] Gamificación. Disponible en: https://www.educativa.com/blog-articulos/gamificacion-el-aprendizaje-divertido/#:~:text=La%20Gamificaci%C3%B3n%20es%20una%20t%C3%A9cnica,concretas%2C%20entre%20otros%20muchos%20objetivos.

[2] Classpip. Disponible en: https://github.com/rocmeseguer/classpip

[3] Moodle. Disponible en: https://docs.moodle.org/all/es/Acerca_de_Moodle

[4] LTI. Disponible en: https://help.blackboard.com/es-es/Learn/Administrator/SaaS/Integrations/Learning_Tools_Interoperability

[5] MongoDB. Disponible en: https://www.mongodb.com/es

[6] MySQL. Disponible en: https://www.mysql.com/

[7] LTIJS. Disponible en: https://www.npmjs.com/package/ltijs

[8] NodeJS. Disponible en: https://nodejs.org/es/about/

[9] Express. Disponible en: https://expressjs.com/es/

[10] Angular. Disponible en: https://angular.io/

[11] Angular Material. Disponible en: https://material.angular.io/

[12] Bootstrap. Disponible en: https://getbootstrap.com/docs/4.5/getting-started/introduction/

[13] Ionic 2. Disponible en: https://desarrolloweb.com/articulos/que-es-ionic2.html#:~:text=Ionic%202%20es%20un%20framework,poco%20aplicaciones%20de%20escritorio%20multiplataforma.

[14] Apache Cordova. Disponible en: https://cordova.apache.org/docs/en/latest/guide/overview/index.html

[15] API REST. Disponible en: https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos

[16] Strongloop. Disponible en: https://strongloop.com/

[17] Endpoint. Disponible en: https://admware.es/que-es-un-endpoint/

[18] HTML. Disponible en: https://es.wikipedia.org/wiki/HTML

[19] Visual Studio Code. Disponible en: https://code.visualstudio.com/docs

[20] GitHub. Disponible en: https://github.com/

[21] Loopback-connector-mongodb. Disponible en: https://loopback.io/doc/en/lb3/MongoDB-connector

[22] Loopback-connector-mysql. Disponible en: https://loopback.io/doc/en/lb3/MySQL-connector.html

[23] SQL. Disponible en: https://es.wikipedia.org/wiki/SQL#:~:text=SQL%20(por%20sus%20siglas%20en,de%20bases%20de%20datos%20relacionales.

[24] ACID. Disponible en: https://es.wikipedia.org/wiki/ACID

[25] JSON. Disponible en: https://www.json.org/json-en.html

[26] Memoria RAM. Disponible en: https://hardzone.es/reportajes/que-es/memoria-ram-pc/

[27] Postman. Disponible en: https://www.postman.com/

[28] MemTest64. Disponible en: https://www.techpowerup.com/memtest64/

[29] BSON. Disponible en: https://en.wikipedia.org/wiki/BSON

[30] IMS Global Consorsium. Disponible en: https://es.wikipedia.org/wiki/IMS_Global

2. Guía paso a paso para el uso de mongodb como BBDD para modelos de datos

Esta guía sirve como referencia para poder configurar nuestra aplicación para que utilice mongodb como base de datos.

Requisitos previos:

1. Tener un **editor de código fuente** (en nuestro caso *Visual Studio Code*)

2. Tener nodejs instalado (https://nodejs.org/es/download/)

Los pasos a seguir son los siguientes:

1. Primero de todo instalaremos mongoDB Community Server a través de:

https://www.mongodb.com/try/download/community

Una vez instalado, recomiendo instalar MongoDB Compass o Shell

2. Instalamos *MongoDB Shell*** mediante:

https://www.mongodb.com/try/download/tools

3. Creamos una nueva BBDD en local (la llamaremos ***test_mongo***). No es necesario configurar una contraseña.

Tras ejecutar el archivo descargado ***mongosh.exe,*** escribimos, en la consola que se nos habrá abierto, lo siguiente:

```
> use test_mongo
> db.test.insert({test:'test1'})
```

4. Creamos una nueva carpeta usando la consola de nuestro editor de código fuente:

```
PS C:\Users\nombreDeUsuario\Escritorio\ mkdir myapp
PS C:\Users\nombreDeUsuario\Escritorio\ cd myapp
```

5. Instalamos *loopback* y comenzamos con la creación del proyecto:

```
PS C:\Users\nombreDeUsuario\Escritorio\myapp npm install -g loopback-cli
PS C:\Users\nombreDeUsuario\Escritorio\myapp lb datasources
```

Nos pedirá lo siguiente:

- Nombre del directorio para el proyecto: **datasources2**
- Versión de *loopback* a utilizar: **escogemos la 3.x**
- Tipo de aplicación: **para esta prueba escogemos un *hello-world***

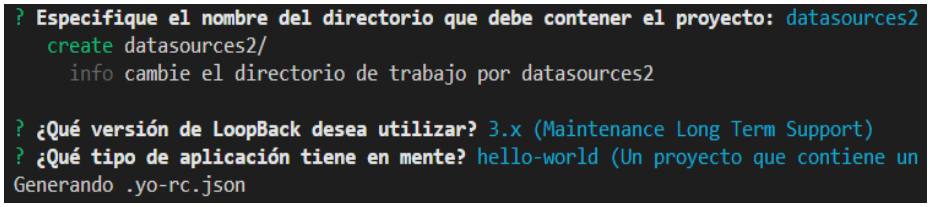

Fig. 2.1 Ejemplo de configuración de la aplicación de prueba

6. Nos cambiamos a la nueva carpeta creada:

PS C:\Users\nombreDeUsuario\Escritorio\myapp cd datasources2

7. Creamos un nuevo *datasource*:

PS C:\Users\nombreDeUsuario\Escritorio\myapp\datasources2 lb datasource
Introducimos los datos siguientes:

- Nombre del datasource: **dbmongo** (o cualquier nombre que te sirva para diferenciar la base de datos)
- Conector: **MongoDB**
- La connection string url: **(pulsamos ENTER)**
- Host: **localhost**
- Port: **27017**
- User y Password: **(pulsamos ENTER)**
- Database: **introducimos el nombre de la BBDD creada en el paso 2 (*test_mongo*)**
- Feature supported by MongoDB v3.1.0 and above: **Yes**
- Instalar loopback-connector-mongodb: **Yes**

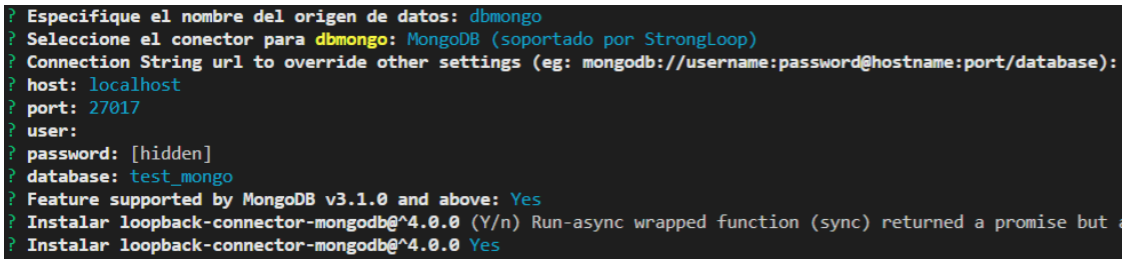

Fig. 2.2 Ejemplo de configuración del datasource

Se nos habrá añadido el nuevo *datasource* en el fichero ***datasources.json*** de nuestra carpeta del proyecto.

8. Creamos 2 modelos de ejemplo (Alumno y Profesor):

PS **lb model <nombre-del-modelo>**
PS C:\Users\nombreDeUsuario\Escritorio\myapp\datasources2 **lb model Alumno**

Introducimos los siguientes datos:

- Nombre del model: **Alumno**
- Origen de datos: **dbmongo** (este el Nombre del datasource creado en el paso anterior)
- Clase base del modelo: **PersistedModel**
- ¿Exponer Alumno mediante API REST? **Yes** (esto nos permitirá implementar las funciones rest)
- Modelo: **común**

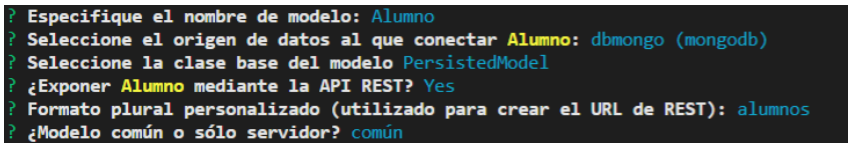

Fig. 2.3 Ejemplo de configuración de un modelo de datos

Ahora introduciremos las propiedades del nuevo modelo Alumno:

- Nombre de la propiedad
- tipo de propiedad
- si es necesaria o no
- valor predeterminado). Esta última propiedad la dejamos vacía (pulsamos ENTER).

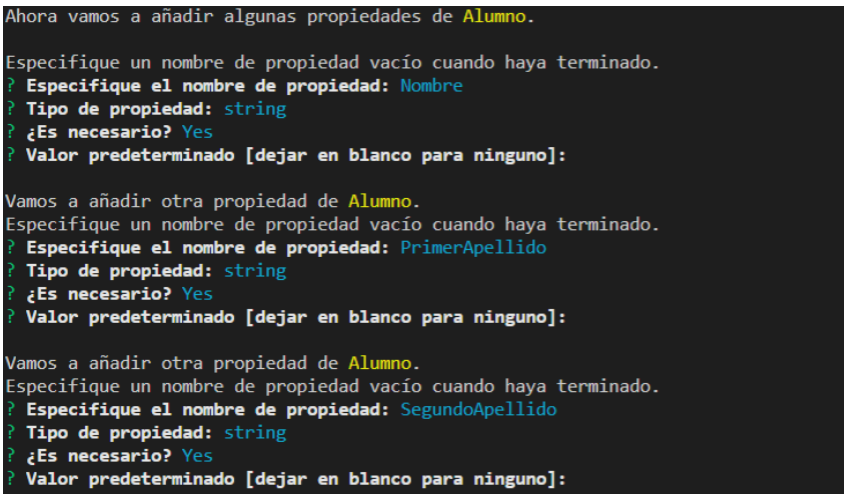

Fig. 2.4 Ejemplo de configuración del modelo de las propiedades de un alumno

Lo mismo para el modelo de Profesor:

PS C:\Users\nombreDeUsuario\Escritorio\myapp\datasources2 **lb model Profesor**

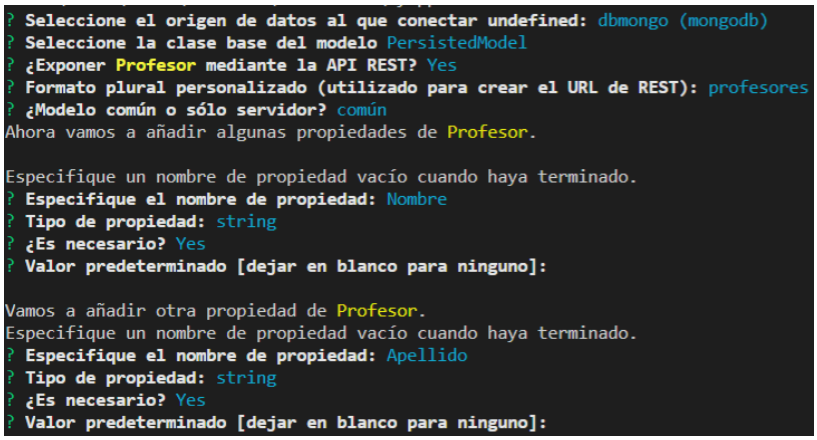

Fig. 2.5 Ejemplo de configuración del modelo de datos de un professor y sus propiedades

9. Creamos una nueva relación entre Profesor y alumno:

PS C:\Users\nombreDeUsuario\Escritorio\myapp\datasources2 **lb relation**
Nos pedirá seleccionar entre diferentes valores:

- En qué modelo queremos crear la relación: **Profesor**
- El tipo de relación: **has many**
- Con qué otro modelo crear la relación: **Alumno**
- El nombre que le queramos dar la relación: **alumnos**
- La clave foránea que queramos usar (esto es el valor que tendremos que introducir en la base de datos para que relacione un modelo con otro): **profesorId**

A lo demás le decimos que no (o pulsamos ENTER).

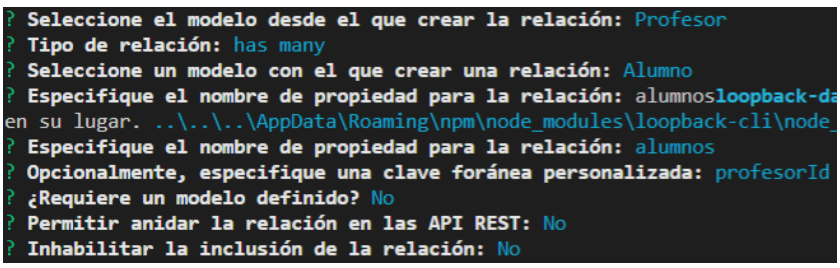

Fig. 2.6 Ejemplo de configuración de las relaciones entre professor-alumno

10. Creamos un nuevo fichero llamado ***automigrate.js*** dentro de la **carpeta *server/boot***:

Asegurándonos que la función *ds.automigrate* queda comentada tras la primera ejecución del paso 11.

En caso contrario es posible que la BBDD se sobrescriba y se pierdan datos.

El contenido del fichero es el siguiente:

```
var server = require('../server');
var fs = require('fs');
// Aquí empieza
var ds = server.dataSources.dbmongo;
ds.automigrate(function(er) {
if (er) throw er;
console.log('Loopback tables created in', ds.adapter.name);
ds.disconnect();
}); // Aquí acaba
module.exports = function(app) {
  'use strict';
  var mongo = app.dataSources.dbmongo;
  console.log('-- Models found:', Object.keys(app.models));
  for (var model in app.models) {
    // eslint-disable-next-line max-len
    console.log('Cheking if table for model ' + model + ' is created and up-to-
date in DB...');
    mongo.isActual(model, function(err, actual) {
      if (!actual) {
        console.log('Difference found! Auto-migrating model ' + model + '...');
        mongo.autoupdate(model, function() {
          console.log('Auto-migrated model ' + model + ' successfully.');
        });
      }
    });
  };
};
```

11. Ejecutamos el proyecto que tenemos mediante el comando siguiente (dentro del terminal dentro de la carpeta del proyecto):

PS C:\Users\nombreDeUsuario\Escritorio\myapp\datasources2 **node .**

Podemos comprobar como en ***mondodb*** se habrán creado las colecciones pertinentes a los modelos que teníamos y en ***localhost:3000/explorer***, podemos ver los diferentes métodos para los modelos creados.

12. Una vez iniciado el proyecto, podemos comentar todo lo que tengamos dentro del fichero llamado ***automigrate.js***. Eso sí, si se realiza alguna modificación en alguno de los modelos, sería necesario volver a cargar este fichero como en el paso 11.

13. En caso de no tener instalado mongoDB se puede dejar la parte asociada del fichero ***datasources.json*** dentro de un fichero .txt. Esto permite que no salten errores y que en cualquier momento puedas recuperar los datos.

Sería simplemente quitar del datasources lo siguiente:

```
 "dbmongo": {
   "host": "localhost",
   "port": 27017,
   "database": "test_mongo",
   "name": "dbmongo",
   "connector": "mongodb"
 },
```

Y guardarlo en un fichero .txt que podemos dejar dentro de la carpeta del proyecto.

14. Para probarlos podemos añadir los diferentes campos usando ***mongodb***:

Para añadir los datos, necesitaremos abrir en un terminal la conexión con mongo.

Si hemos instalado **MongoDB Shell** hacemos clic en el icono descargado.

Para introducir los datos hacemos lo siguiente (dentro de la ventana de cmd que se nos habrá abierto):

Profesor:

```
> use test_mongo
> db.Profesor.insert({Nombre:'Troy',Apellido: 'Caicedo'})
```

Ahora necesitamos saber el id que mongodb ha creado automáticamente. Para ello realizamos:

```
> db.Profesor.find()
```

Copiamos el valor que aparece como **_id** para introducirlo dentro del campo profesorId del Alumno.

Alumno:

```
> db.Alumno.insert({Nombre:'Andrés',PrimerApellido:
  'Caicedo',SegundoApellido: 'Uvidia',profesorId:
  ObjectId('5f0ddbb9aa85250b704e6b95')})
```

Si hemos instalado ***MongoDBCompass*** se nos hará algo más sencilla la visualización e introducción de datos.

Simplemente abrimos la aplicación de ***MongoDBCompass*** y nos conectamos a nuestra base de datos (localhost) . Allí veremos las colecciones que se han creado a partir de los modelos. Entramos en la queramos añadir datos nuevos y le damos al botón de añadir datos > insertar documento:

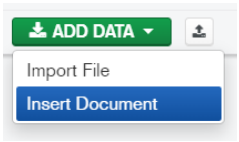

Fig. 2.7 Botón de añadir datos mediante inserción de un documento

Nos aparecerá la siguiente ventana:

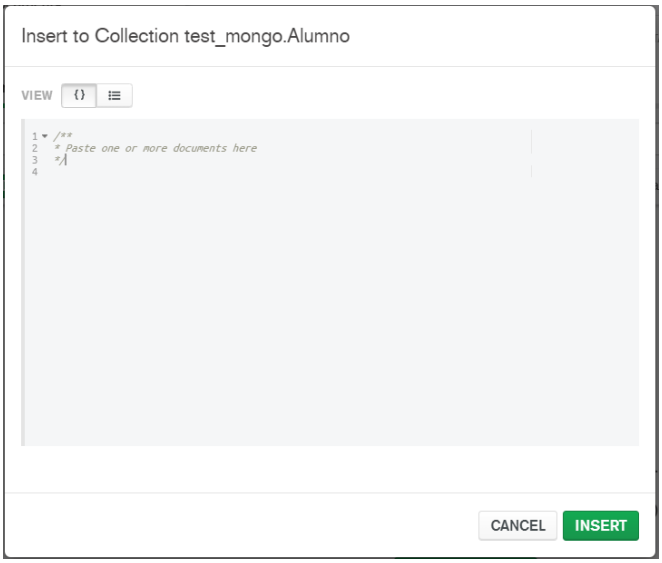

Fig. 2.8 Ventana para introduir manualment una nueva colección

Para la colección de **Profesor** copiamos y pegamos lo siguiente:

{"Nombre" : "Troy","Apellido" : "Caicedo"}

Le damos a ***insert***.

Una vez creado, mogodb le asignará un id automáticamente (del tipo ObjectId). Este valor lo necesitaremos para crear al Alumno. En mi caso tengo el ***ObjectId("5f0ddbb9aa85250b704e6b95").***

Para la colección de Alumno copiamos y pegamos lo siguiente:
{Nombre" : "Andres","PrimerApellido" : "Caicedo","SegundoApellido" : "Uvidia, "profesorId" : { "$oid": "5f0ddbb9aa85250b704e6b95"}}

Le damos a ***insert***.

Si buscamos ***localhost:3000/explorer*** en nuestro navegador favorito y, hacemos un get de los diferentes modelos, podremos ver que recuperamos los datos que hemos introducido.

También los podemos conseguir usando MongoDB Compass o bien haciendo un ***db.Profesor.find()*** y ***db.Alumno.find()*** usando MongoDB Shell (tal y como hemos visto en el paso 11).

1.1. Importar y exportar datos

Importar datos desde un fichero json a mongoDB

Dentro de la carpeta donde tengas mongodb instalado, en mi caso ***C:\Program Files\MongoDB\Server\4.2\bin*** abrimos un terminal (CMD) y ejecutamos la siguiente línea de código:

mongoimport.exe --db [nombre Base de Datos] /c [nombre de la Colección] /file [fichero json a usar] --type json --batchSize 1 –jsonArray

En mi caso, vamos a importar el fichero importTest.json en mi colección JuegoDePuntos y mi base de datos tfg_mongo_loopback:

mongoimport.exe --db tfg_mongo_loopback **/c** JuegoDePuntos **/file** C:\Users\andre\OneDrive\Escritorio\importTest.json **--type** json **--batchSize** 1 **–jsonArray**

Exportar colección de mongodb a fichero json

Dentro de la carpeta donde tengas mongodb instalado, en mi caso ***C:\Program Files\MongoDB\Server\4.2\bin*** abrimos un terminal (CMD) y ejecutamos la siguiente línea de código:

mongoexport.exe --db [nombre Base de Datos] /c [nombre de la Colección] /out [fichero json que queremos crear] –jsonArray /pretty

En mi caso, vamos a exportar, en un fichero llamado exportTest.json ubicado dentro de ***C:\Users\andre\OneDrive\Escritorio***, los datos de mi colección JuegoDePuntos y mi base de datos tfg_mongo_loopback:

mongoexport.exe --db test_tfg_mongo **--jsonArray /c** Alumno **/out** C:\Users\andre\OneDrive\Escritorio\exportTest.json **/pretty**

Para exportar e importar una base de datos en su totalidad, podemos realizar lo siguiente (en una ventana cmd dentro de la carpeta donde tengamos mongoDB instalado):

Exportar basededatos:

C:\Program Files\MongoDB\Server\4.2\bin> mongodump.exe --archive="C://Users//andre//OneDrive//Escritorio//testmongodb" --db=test_mysql

Importar basededatos

C:\Program Files\MongoDB\Server\4.2\bin> mongorestore --archive=" C://Users//andre//OneDrive//Escritorio//testmongodb" --db=test_mysql

3. Guía paso a paso para el uso de MySQL como BBDD para modelos de datos

Esta guía sirve como referencia para poder configurar nuestra aplicación para que utilice mysql como base de datos.

Requisitos previos:

1. Tener un **editor de código fuente** (en nuestro caso *Visual Studio Code*)
2. Tener nodejs instalado (https://nodejs.org/es/download/)

Pasos a seguir:

1. Instalamos ***mysql*** (en nuestro caso la versión gratuita *Community Server 8.0)*:

   https://dev.mysql.com/downloads/mysql/

2. Creamos una nueva BBDD en local (la llamaremos test_mysql).

3. Para conectarnos a mysql necesitamos acceder o bien desde la carpeta de la instalación o bien accediendo desde el terminal (cmd) y llamando a mysql:

   mysql -u root -p

4. Creamos una nueva carpeta usando la consola de nuestro editor de código fuente:

   PS C:\Users\nombreDeUsuario\Escritorio\ mkdir myapp
   PS C:\Users\nombreDeUsuario\Escritorio\ cd myapp

5. Instalamos loopback y comenzamos con la creación del proyecto:

   PS C:\Users\nombreDeUsuario\Escritorio\myapp **npm install -g loopback-cli**
   PS C:\Users\nombreDeUsuario\Escritorio\myapp **lb datasources**

   Nos pedirá lo siguiente:

   - Nombre del directorio para el proyecto: **datasources2**
   - Versión de *loopback* a utilizar: **escogemos la 3.x**
   - Tipo de aplicación: **para esta prueba escogemos un *hello-world***

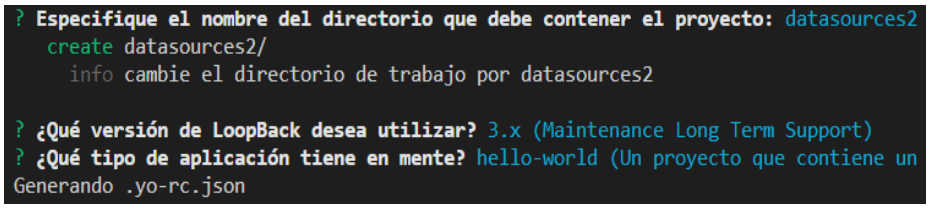

Fig. 3.1 Ejemplo de creación de una aplicación

6. Nos cambiamos a la nueva carpeta creada:

PS C:\Users\nombreDeUsuario\Escritorio\myapp cd datasources2

7. Creamos un nuevo *datasource*:

PS C:\Users\nombreDeUsuario\Escritorio\myapp\datasources2 lb datasource

Introducimos los datos siguientes:

- Nombre del datasource: **mysql** (o cualquier nombre que te sirva para diferenciar la base de datos)
- Conector: **MySQL**
- La connection string url: **(pulsamos ENTER)**
- Host: **localhost**
- Port: **3306**
- User: **root**
- Password: **contraseña escogida al instalar mysql**
- Database: **introducimos el nombre de la BBDD creada en el paso 3 (*test_mysql*)**
- Feature supported by mysql v3.1.0 and above: **Yes**
- Instalar loopback-connector-mysql: **Yes**

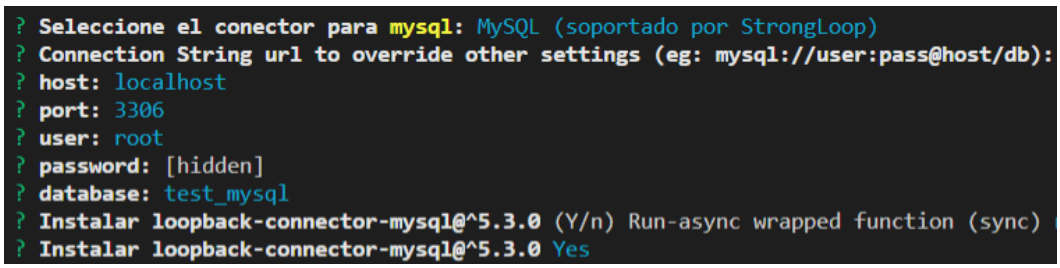

Fig. 3.2 Ejemplo de creación de MySQL como datasource

Se nos habrá añadido el nuevo *datasource* en el fichero ***datasources.json*** de nuestra carpeta del proyecto.

8. Creamos 2 modelos de ejemplo (Alumno y Profesor):

PS **lb model <nombre-del-modelo>**
PS C:\Users\nombreDeUsuario\Escritorio\myapp\datasources2 **lb model Alumno**

Introducimos los siguientes datos:

- Datos al que conectar: **mysql (mysql)**
- Nombre del model: **Alumno**
- Origen de datos: **mysql** (este el Nombre del datasource creado en el paso anterior)
- Clase base del modelo: **PersistedModel**
- ¿Exponer Alumno mediante API REST? **Yes** (esto nos permitirá implementar las funciones rest)
- Modelo: **común**

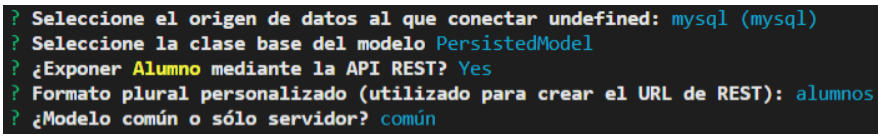

Fig. 3.3 Ejemplo de creación del modelo de alumno

Ahora introduciremos las propiedades del nuevo modelo Alumno (nombre de la propiedad, tipo de propiedad, si es necesaria o no y valor predeterminado).

Esta última propiedad la dejamos vacía (pulsamos ENTER).

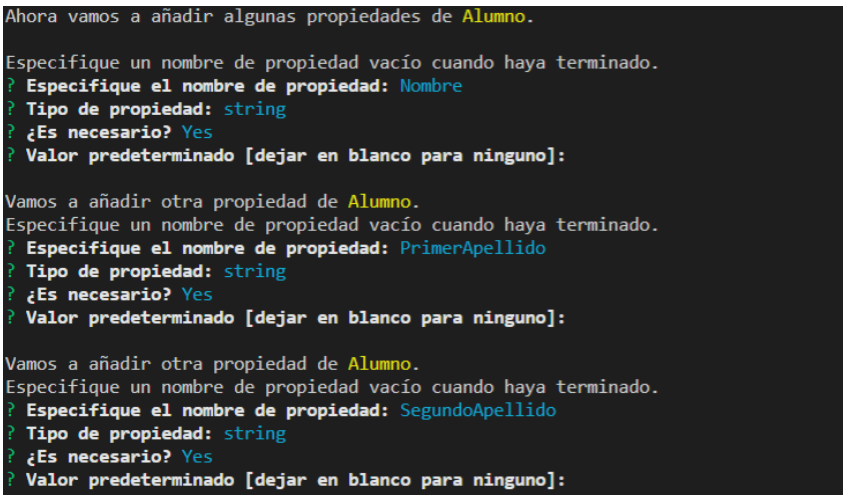

Fig. 3.4 Ejemplo de creación de las propiedades del modelo de alumno

Lo mismo para el modelo de Profesor:

PS C:\Users\nombreDeUsuario\Escritorio\myapp\datasources2 **lb model Profesor**

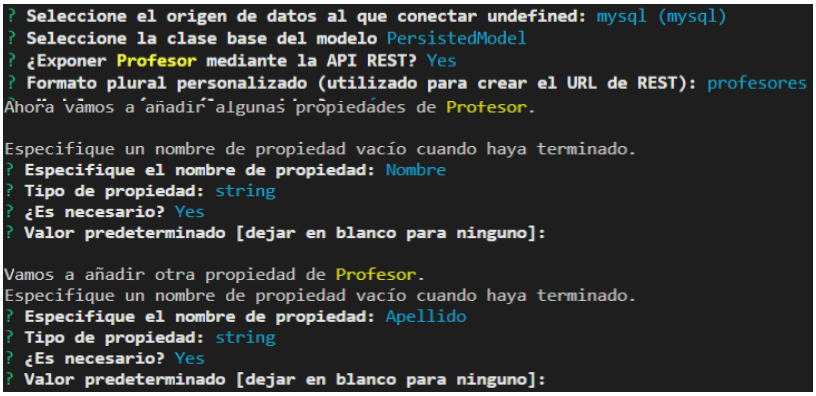

Fig. 3.5 Ejemplo de creación del modelo professor y sus propiedades

9. Creamos una nueva relación entre Profesor y alumno:

PS C:\Users\nombreDeUsuario\Escritorio\myapp\datasources2 **lb relation**
Nos pedirá seleccionar entre diferentes valores:

- En qué modelo queremos crear la relación: **Profesor**
- El tipo de relación: **has many**
- Con qué otro modelo crear la relación: **Alumno**
- El nombre que le queramos dar la relación: **alumnos**
- La clave foránea que queramos usar (esto es el valor que tendremos que introducir en la base de datos para que relacione un modelo con otro): **profesorId**

A lo demás le decimos que no (o pulsamos ENTER).

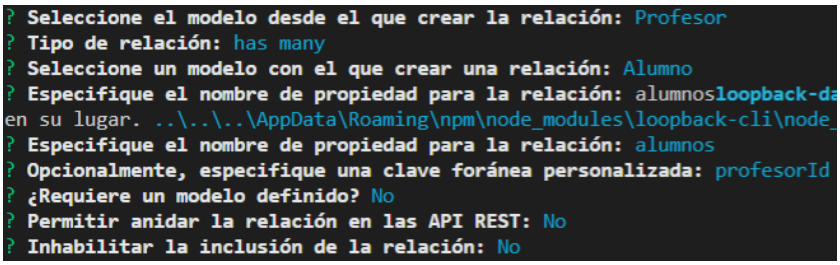

Fig. 3.6 Ejemplo de creación de la relación professor-alumno

10. Creamos un nuevo fichero llamado ***automigrate.js*** dentro de la **carpeta *server/boot*** con el siguiente contenido:

Asegurándonos que la función *automigrate* queda comentada tras la primera ejecución del paso 11.

En caso contrario es posible que la BBDD se sobrescriba y se pierdan datos.

```
/* eslint-disable strict */
module.exports = function(app) {
  // comment this function after 1st run to prevent loss of data
  /*
  app.dataSources.mysql.automigrate();
  console.log('Performed automigration.');
  //
  */
  app.dataSources.mysql.autoupdate();
  console.log('Performed autoupdate.');
};
```

11. Ejecutamos el proyecto que tenemos mediante el comando siguiente (dentro del terminal dentro de la carpeta del proyecto):

PS C:\Users\nombreDeUsuario\Escritorio\myapp\datasources2 **node .**

Podemos comprobar como en ***mysql*** se habrán creado las colecciones pertinentes a los modelos que teníamos y en ***localhost:3000/explorer***, podemos ver los diferentes métodos para los modelos creados.

12. Una vez iniciado el proyecto, podemos comentar todo lo que tengamos dentro del fichero llamado ***automigrate.js***. Eso sí, si se realiza alguna modificación en alguno de los modelos, sería necesario volver a cargar este fichero como en el paso 11.

13. Para probarlos podemos añadir los diferentes campos usando ***mysql***:

Para añadir los datos, necesitaremos abrir, en un terminal, la conexión con ***mysql***:

C:\Users\nombreDeUsuario\Escritorio > mysql -u root -p

Deberemos introducir la contraseña creada en el paso 1

Para introducir los datos hacemos lo siguiente:

Profesor:

```
> use test_mysql
> INSERT INTO profesor VALUES(1,'Andrés', 'Caicedo');
```

Alumno:

```
> INSERT INTO alumno VALUES(1,'Andrés', 'Caicedo', 'Uvidia', 1);
```

Si buscamos ***localhost:3000/explorer*** en nuestro navegador favorito y, hacemos un *get* de los diferentes modelos, podremos ver que recuperamos los datos que hemos introducido.

3.1. Importar y exportar base de datos

Si quiero exportar una base de datos para luego importarla en otro servidor desde un terminal:

C:\Program Files\MySQL\MySQL Server 8.0\bin> mysqldump –u usuario -p nombre_basededatos > fichero_exportación.sql

Para importar los datos:

C:\Program Files\MySQL\MySQL Server 8.0\bin> mysql –u usuario -p nombre_basededatos < fichero_importación.sql

Si solo queremos una tabla en concreto:

C:\Program Files\MySQL\MySQL Server 8.0\bin> mysqldump –u usuario -p nombre_basededatos nombre_tabla > fichero_exportación.sql

3.2. Importar JSON en MySQL

Para poder importar datos desde un fichero .json tenemos diferentes formas de realizarlo.

3.2.1. MySQL Shell

Usando la utilidad MySQL Shell podremos importar un fichero JSON en una tabla o colección de MySQL.

Primero tenemos que activar el protocolo mysqlX. Vamos a la carpeta donde tengamos el servidor de MySQL y dentro del bin ejecutamos lo siguiente en el cmd:

C:\Program Files\MySQL\MySQL Server 8.0\bin> **mysqlsh -u root -h localhost --mysql --dba enableXProtocol**

Conectamos con el servidor MySQL usando MySQLShell:

C:\Program Files\MySQL\MySQL Server 8.0\bin> **mysqlsh -u root -h localhost –mysqlx**

Suponiendo que queremos importar el fichero .json llamado testMySQL.json en la tabla con nombre alumno de la base de datos *test_mysql***,** el comando a ejecutar sería:

MySQL Shell > **util.importJson("testMySQL.json", {schema: "*test_mysql*", table: "alumno"});**

3.2.2. MySQL Workbench

En este caso la importación es mucho más sencilla. Basta entrar en la base de datos, ir a cualquier tabla y hacer clic en importar. Elegiremos el fichero, escogeremos donde lo queremos importar y dónde va cada valor dentro de la tabla.

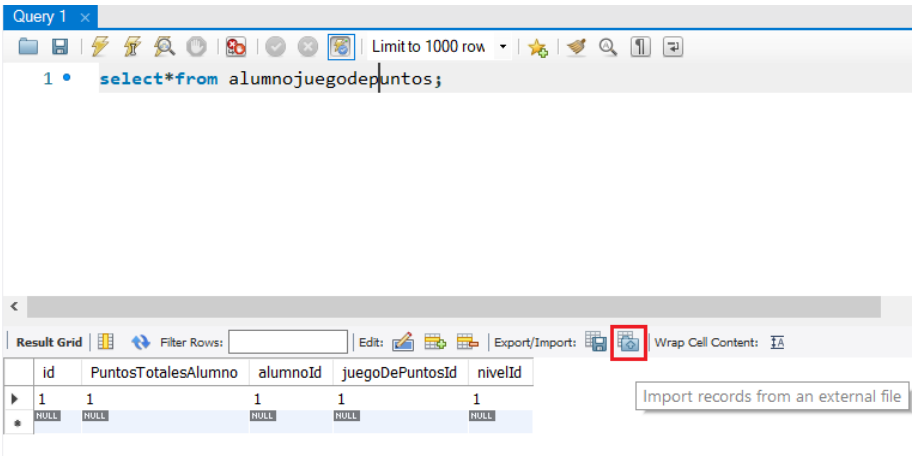

Fig. 3.2.2.1 Importar datos desde un fichero externo

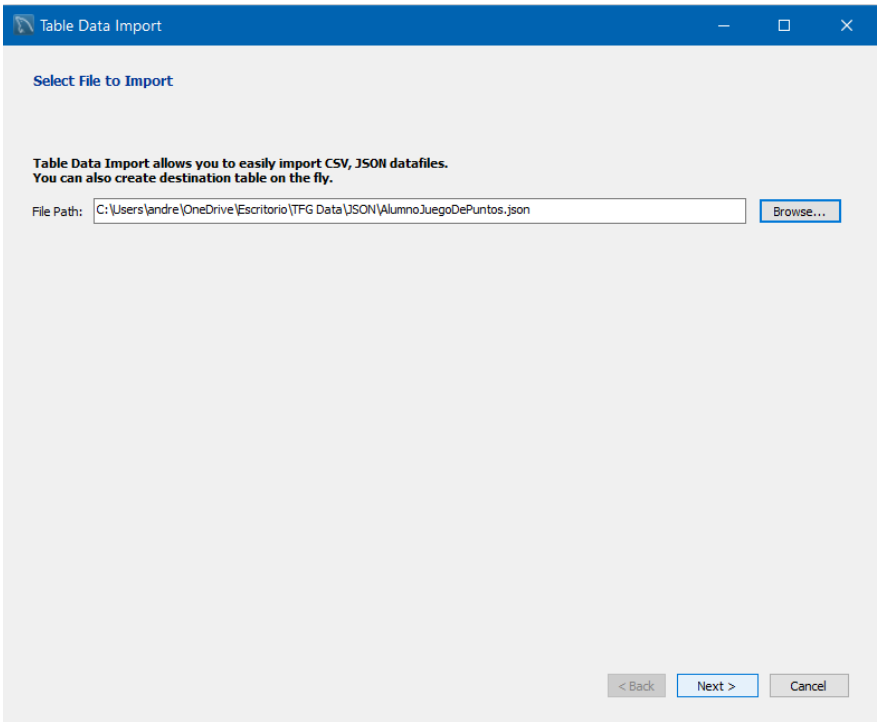

Fig. 3.2.2.2 Seleccionar ruta del fichero a exportar

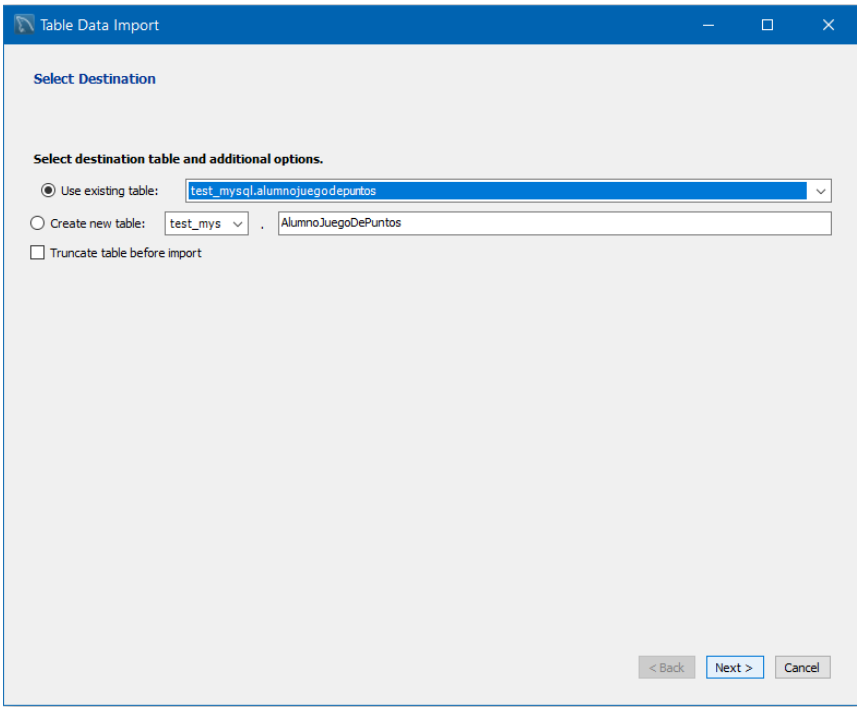

Fig. 3.2.2.3 Escoger dónde importar los datos

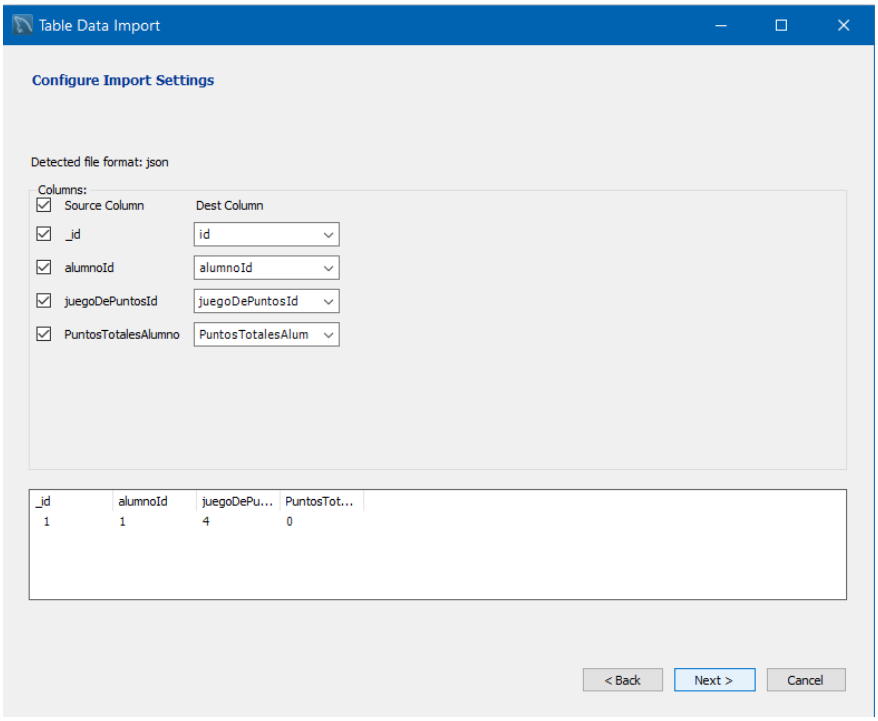

Fig. 3.2.2.3 Escoger columnes para cada dato del fichero

3.3. Exportar tabla a formato .json

Para exportar una tabla a formato json tenemos dos opciones:

- Usando MySQL Workbench
- Usar un script en PHP que lo hace por ti:

  https://github.com/paolobertani/Dump-table-to-JSON

En el primer caso es tan fácil como entrar en la base de datos, ir a la tabla que queramos exportar y hacer clic en exportar. Elegiremos el formato json y listos.

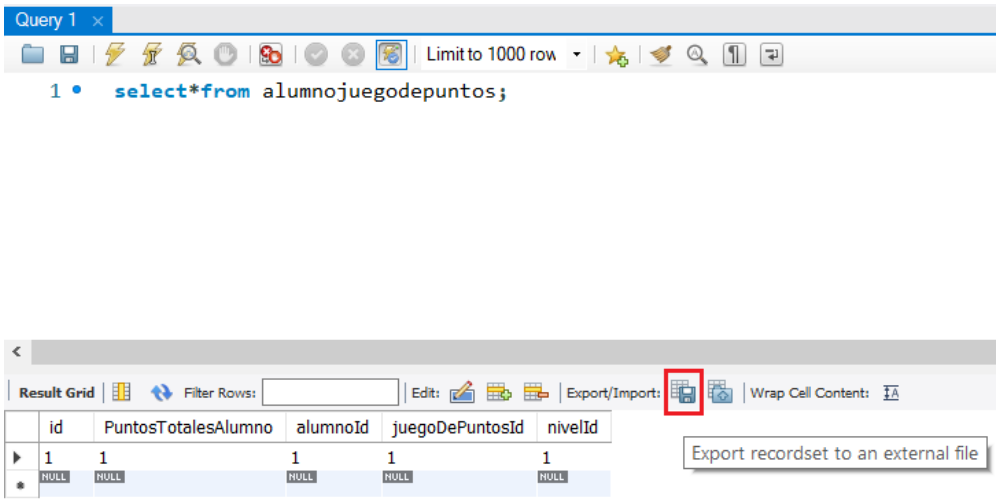

Fig. 3.3.1 Exportar tabla a fichero externo

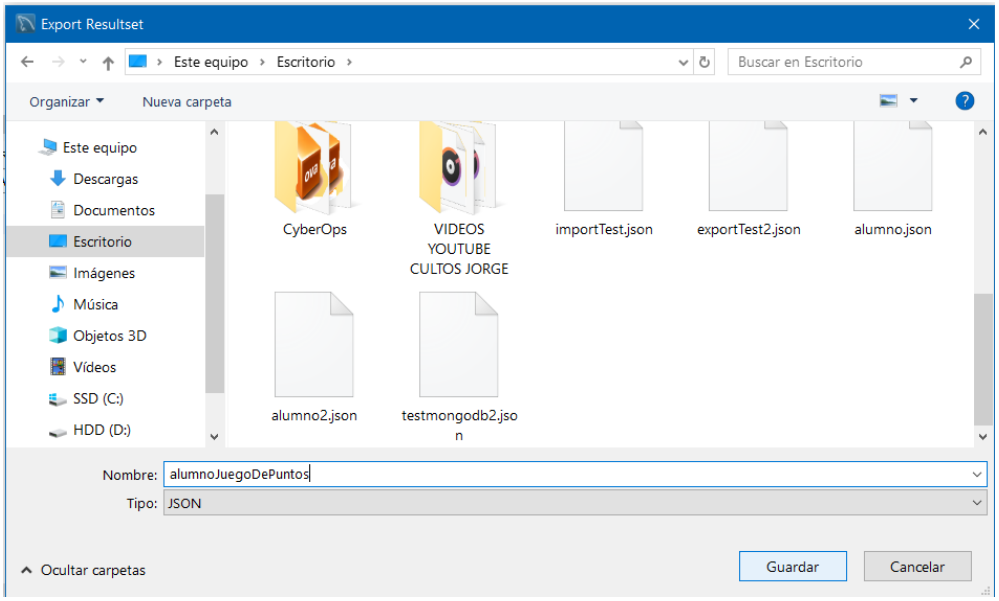

Fig. 3.3.2 Escoger ubicación y nombre del fichero json

4. Guía para instalar y poner en marcha Moodle y LTIJS

Esta guia sirve para ver los pasos a seguir para implementar y poner en marcha LTIJS utilizando Moodle.

4.4. Instalación y puesta en marcha de Moodle

Lo primero es ir a la página oficial de Moodle:

https://docs.moodle.org/25/en/Complete_install_packages_for_Windows

Tras esto, seguimos los pasos que encontramos en la página y descargamos todo lo necesario. Siguiendo el tutorial descrito no habrá ningún tipo de dificultad instalando Moodle.

En cuanto a la puesta en marcha, simplemente accedemos a la carpeta donde tengamos instalado Moodle y hacemos clic en la aplicación ***Start Moodle***. Para cerrar el servidor es recomendable utilizar la aplicación ***Stop Moodle***, para evitar posibles problemas y asegurarnos que se detiene por completo.

Una vez puesto en marcha nuestro Moodle local, tendremos que esperar a tener la configuración de LTI en la aplicación para poder proseguir.

4.5. Instalación y puesta en marcha de LTIJS

Para poder hacer uso de LTI he utilizado la librería LTIJS, la cual nos permite implementar la última versión de LTI, la 1.3.

Primero de todo instalamos *LTIJS* y *dotenv* mediante el siguiente comando en la carpeta del proyecto (en mi caso dentro del servidor de Classpip):

npm install ltijs dotenv

Tras haber instalado correctamente las librerías necesarias, podemos empezar con la configuración del Servidor.

Necesitamos crear variables globales que harán falta:

```
// Requiring LTIJS provider
const Lti = require('ltijs').Provider
// Requiring path
const path = require('path')
// Loading environment variables
require('dotenv').config()
```

Una vez configuradas las variables globales, podemos crear la instancia del objeto LTI:

```
const lti = new Lti(process.env.LTI_KEY,
    // Setting up database configurations
  {url: 'mongodb://' + process.env.DB_HOST + '/' + process.env.DB_DATABASE,
    connection: {user: process.env.DB_USER, pass: process.env.DB_PASS}},
    {appUrl: '/start', loginUrl: '/login', logger: true});
```

Con tal de guardar la configuración de la base de datos y la clave LTI, tendremos que crear un fichero .env que contenga algo similar a lo siguiente:

DB_HOST=localhost
DB_DATABASE=ltimoodle
LTI_KEY=LTIKEY
MOODLE_URL=http://localhost/

En caso de tener usuario y contraseña establecidos en la base de datos, se tendría que añadir dos campos más: DB_USER y DB_PASS.

Ahora necesitamos acceder al apartado de configuración de herramienta externa de Moodle.

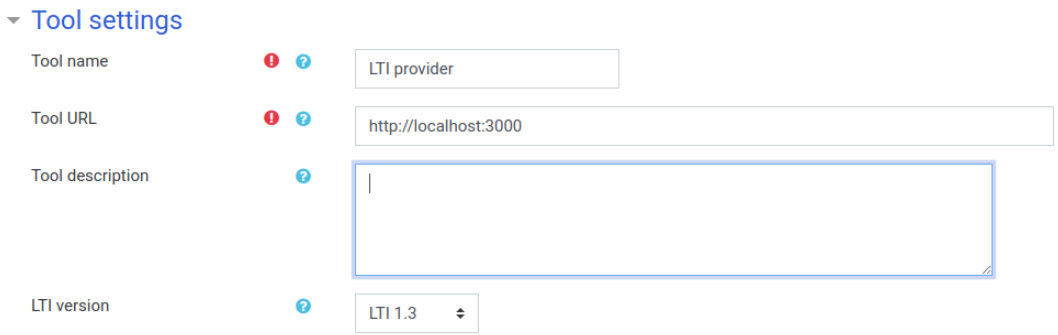

Fig 4.5.1 Configuración de herramienta externa en Moodle I

Tal y como se puede apreciar en **la Fig 4.5.1**, ahora es momento de establecer el nombre de la herramienta (según nos convenga), la URL de la herramienta (en este caso http://localhost:3000) y seleccionamos la versión 1.3 de LTI.

Tras esto, es turno de añadir la configuración de rutas. He utilizado /login como ruta de login, como URI de redirección uso la de http://localhost:3000/start, como herramienta preconfigurada y contenedor por defecto embebido. Todo esto queda tal y como vemos en la **Fig 4.5.2.**

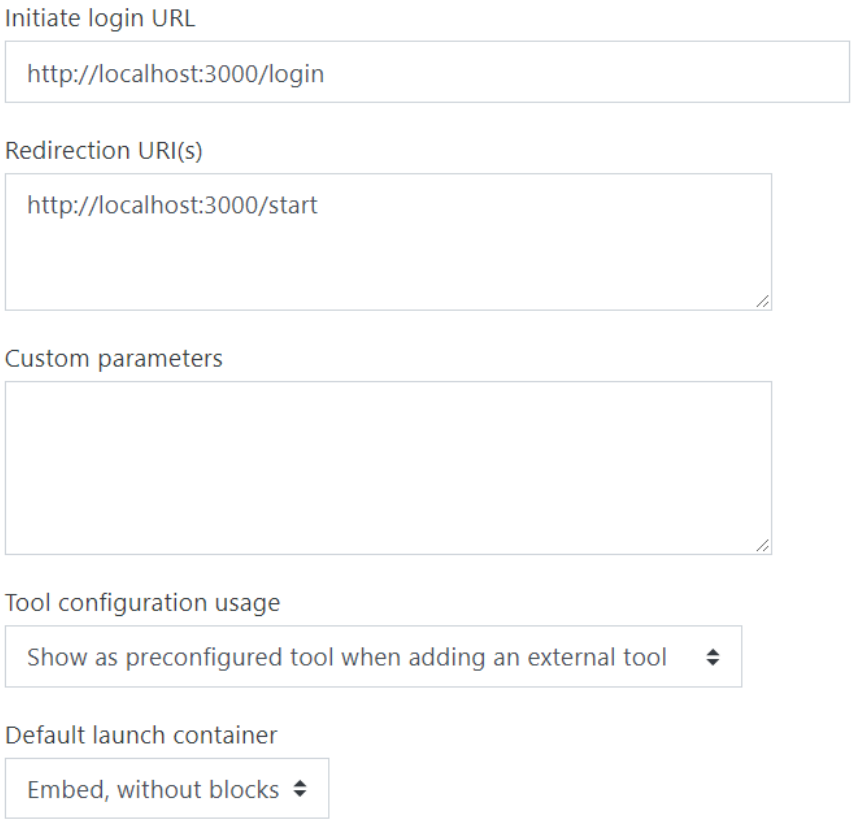

Fig 4.5.2 Configuración de herramienta externa en Moodle II

Ahora establezco los permisos de privacidad y los servicios, para que podamos acceder a información importante de la herramienta tal y como podemos observar en la **Fig 4.5.3**.

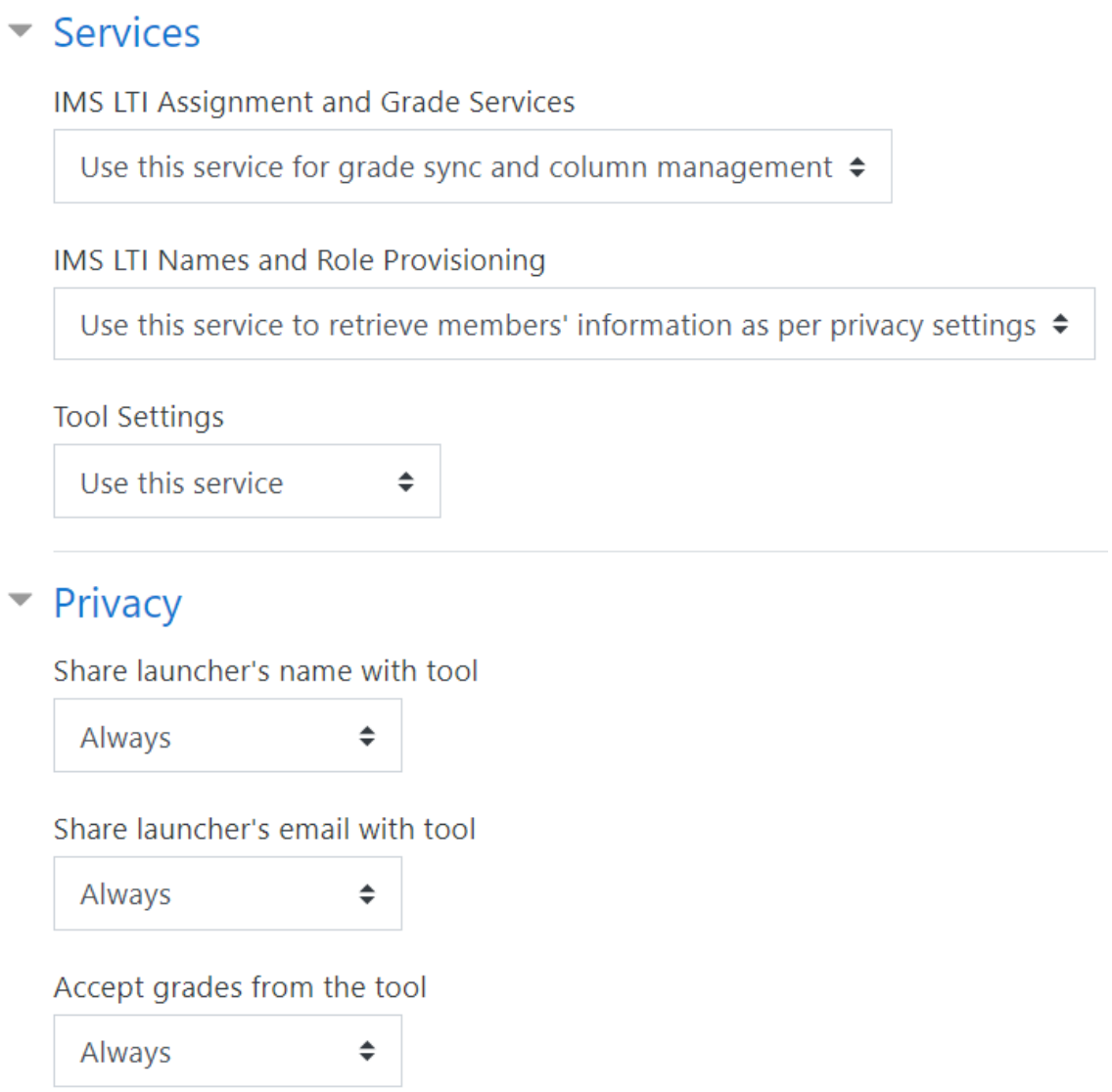

Fig 4.5.3 Configuración de herramienta externa en Moodle III

El resto de campos los dejamos vacíos mientras acabamos de configurar el Servidor. Guardamos la nueva herramienta que hemos creado.

Al hacer clic en el primer icono (marcado en un cuadro rojo en la **Fig 4.5.4**) podremos obtener la información necesaria para registrar la plataforma en el Servidor (**Fig 4.5.5**).

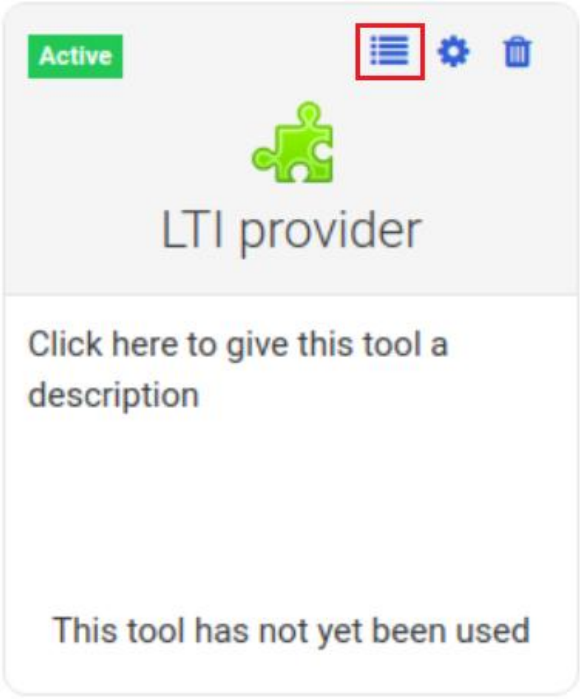

Fig 4.5.4 Herramienta externa en Moodle

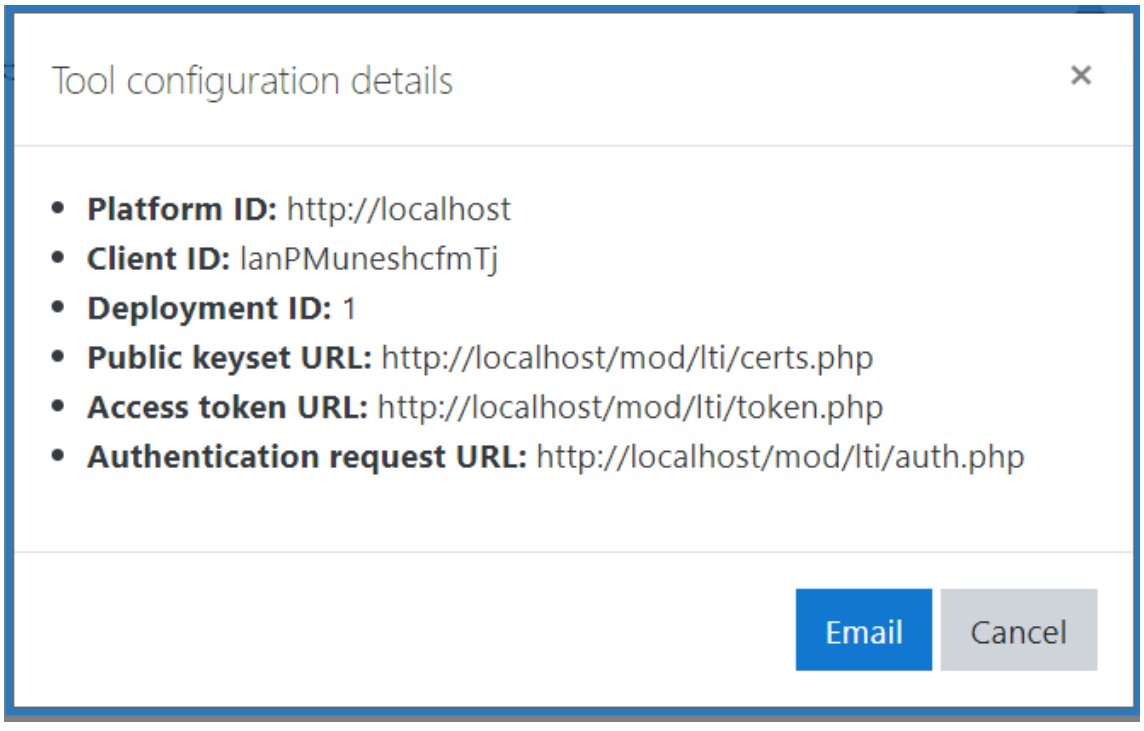

Fig 4.5.5 Detalles de configuración de la herramienta externa en Moodle

Ahora es momento de asociar la plataforma a la aplicación:

```
async function setup() {
```

```
// Deploying provider, connecting to the database and starting e
xpress server.
await lti.deploy({serverless: true});
// Register Moodle as a platform
const plat = await lti.registerPlatform({
url: 'http://localhost',
name: 'Local Moodle',
clientId: 'lanPMuneshcfmTj',
authenticationEndpoint: 'http://localhost/mod/lti/auth.php',
accesstokenEndpoint: 'http://localhost/mod/lti/token.php',
authConfig: {
method: 'JWK_SET',
key: 'http://localhost/mod/lti/certs.php'},
});
```

Simplemente copiamos los datos obtenidos de la **Fig 4.5.5** y los introducimos en los campos correspondientes.

Es importante remarcar que, al ya tener un servidor ejecutado no necesitamos crear uno nuevo, por lo que haremos uso de la opción sin servidor de LTI (serverless: true). Tras esto, se registra la plataforma y, una vez finalizado el registro, se imprime en consola la clave de la plataforma. Esto nos será necesario para acabar de configurar la conexión LTI entre el servidor y Moodle.

Finalmente, se tiene que crear una función que se llama cada vez que hay un lanzamiento con éxito:

```
  console.log(await plat.platformPublicKey());
  lti.onConnect((connection, request, response) => {
    // Set the rerouting towards /begin
    lti.redirect(response, '/begin', {ignoreRoot: true,
     isNewResource: true});
    console.log('lti.onconnect okay\n');
  });
}
```

Con esto, tras un inicio exitoso, seremos redirigidos a la página */begin* en la que se pasa el parámetro ltik (un token JWT firmado que sirve para validar las peticiones del proveedor). Por tanto, la ruta pasará a ser ***/begin?ltik=KEY***.

Ahora simplemente se tiene que configurar las rutas a utilizar, por el momento solo se hará uso de la ruta /begin:

```
  app.get('/begin', lti.app, async (req, res) => {
    return res.send('It works!');
```

```
});
```

Ejecutamos el proyecto mediante el comando npm run start y observaremos que obtenemos una clave pública (**Fig 4.5.6**).

```
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEA0UQ9EnEppIzlSyBBgjhv
G31kDnqUVPB40ca4siK+vI3n08Zl5TwcTyed6WZXlt4pKAbGxehvRkL+CmzkYs7o
9LuPMXC+/otyryBGClt1xcllLekcvPR857W/xZCRPUNvU6nCO8TPEupAdCMscav4
2dMlM6qz2zwWg3W+4xq6YZOkrmUxWjppqREQ+dbcuCuEsIDY7qaYjr665aSFvJG+
C4DUO+ZEM0pb85D5PmT4DBAVyPfdDRuk33/skTR1ZtuAvntF0jHbsTpDHJGC1Bqf
BgKq7VEGwv0crfpRrd8yIiGxZkFEK2Uf1ATdtvtbGw+God1G8/K7dYGA1cPTkvcw
GsQWMZaTv/aCmH9nVaQy0eObNetyr14SPtCZFF1al8I+1763zgonngkWg6TocpfR
LNWAh57qQh296cWJKktYJ+hGegZH4QHQA3EX5ybpSViwpa3viviLqBLSx/GzkVpu
kti96imkeagTjo0S397SBZRvC/7S7agP1HUw9ZueMz4zXMnqQEsNVxLkICGAl7nK
W638srfSPr2Vv/ea0sfzvuENiddP4bJic34il7kkrymdszQZ4TKsoDh11tg2jHsF
kj9nsomRQOi4fT8MPuB1rgOCfiRCn66kIUBMkaXfpbVgFAnb1MX9F2LNcfcYWCoG
At85ztCXrGRMTSPUBewIpW8CAwEAAQ==
-----END PUBLIC KEY-----
```

Fig 4.5.6 Clave pública de LTI (servidor)

Esta clave la tendremos que poner en el campo de clave pública dentro de la ventana de configuración de la herramienta externa de Moodle (**Fig 6.3.2.7**). Para que aparezca esta opción, tenemos que seleccionar la Clave RSA como tipo de clave pública.

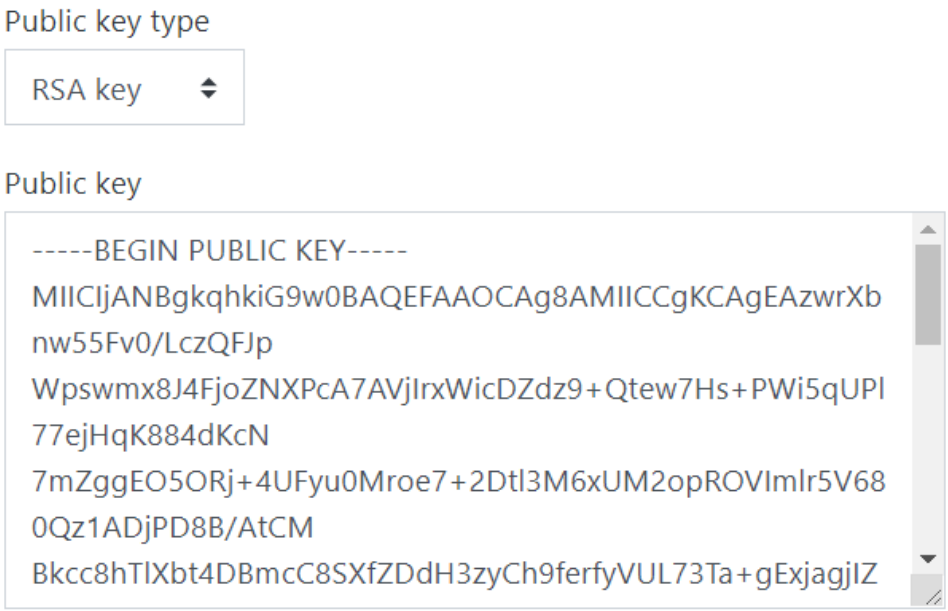

Fig 4.5.7 Configuración de herramienta externa en Moodle IV

Después de guardar los cambios efectuados, solo nos queda crear una nueva actividad (**Fig 4.5.8**).

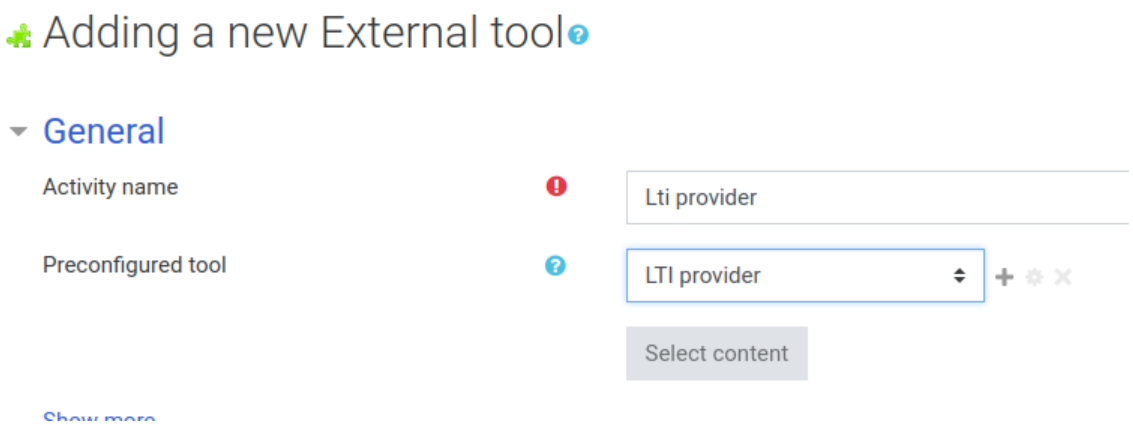

Fig 4.5.8 Configuración de actividad en Moodle

Con la configuración realizada, al entrar en la actividad creada, veremos una ventana dentro de la misma página de la actividad que nos mostrará lo que podemos observar en la **Fig 4.5.9**.

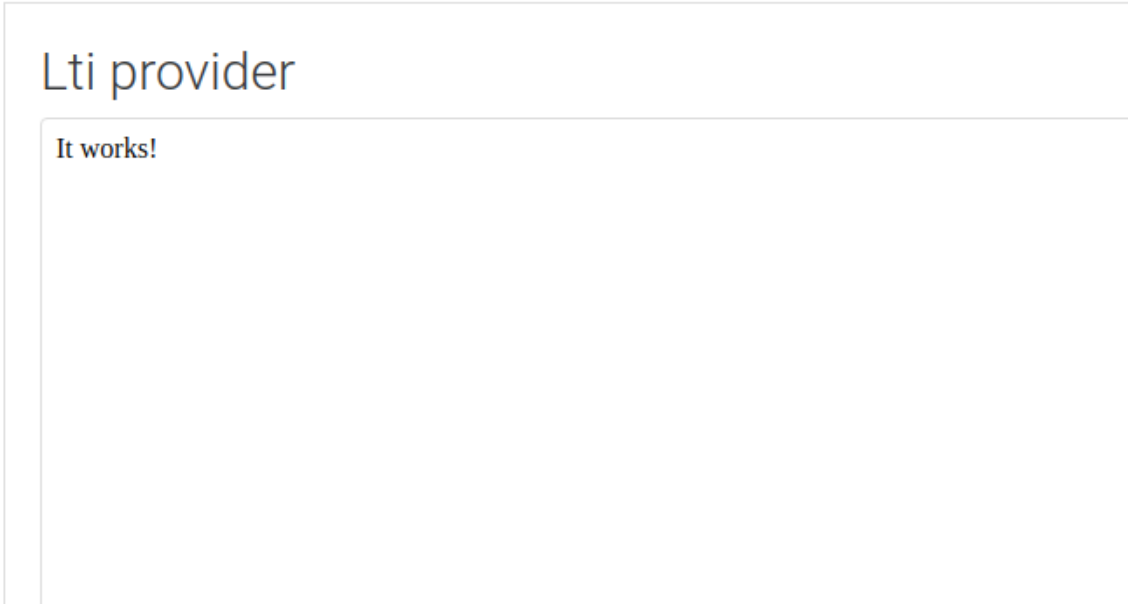

Fig 4.5.9 Resultado de la actividad de Moodle

5. Guía para implementar el Inicio de sesión

Primero se añade el campo *UsuarioMoodle* con parámetro por defecto vacío: ‘’, dentro del Modelo del Alumno (podemos ver un ejemplo en el **Anexo 6.2**).

Ahora es turno de instalar las librerías necesarias. Esto lo podemos hacer mediante el siguiente comando: ***npm install babel-eslint ejs***.

Una vez instaladas las librerías configuramos el servidor para que utilice ejs para renderizar páginas.

```
var engine = require('ejs-mate');
var bodyParser = require('body-parser');
module.exports = async function(app) {
  app.use(bodyParser.urlencoded({extended: false}));
  app.use(bodyParser.json());
  app.engine('ejs', engine);
  app.set('view engine', 'ejs');
  app.set('views', path.join(__dirname, '../views'));
  // Logic when rerouted to /begin
  app.get('/begin', lti.app, async (req, res) => {
    // Render with ejs the login page
 registeredUser(nameUser, passUser).subscribe(registration => {
      miRegisteredUser = registration;
      let userData = {username: nameUser, password: passUser, re
gistered: miRegisteredUser, classpipRegister: true};
      res.render('login.ejs', {data: userData});
    });
  });
```

Utilizo el *bodyParser* para parsear las respuestas, definimos que se utilizará el formato *json* (ya que las respuestas serán en este formato), establecemos *ejs* como motor del servidor, le indicamos el motor de vista y la ruta de la carpeta donde guardaremos las páginas que queramos mostrar.

Tras esto, se crean una variable para guardas los datos necesarios para el inicio de sesión (nombre de usuario y la contraseña, obtenidas a partir de los datos que facilita la plataforma de Moodle). También buscamos en la base de datos, a partir de los datos cogidos de la plataforma, si el usuario ya ha sido registrado en la plataforma o no. Esto último nos devuelve un id o un string vacío. Una vez se tienen los datos, se puede indicar que se quiere renderizar la página de *login.ejs* y que le pasaremos los datos obtenidos con anterioridad.

Se puede ver una función que extrae los datos de la base de datos, ***isRegistered(nameUser, passUser)***. Dicha función se implementa de la siguiente manera:

```
function registeredUser(nombreUser, passwordUser) {
  const registeredObservable = new Observable(obs => {
    let registered = [];
    request.get(server + '/api/Alumnos?filter[where][Nombre]=' +
    nombreUser + '&filter[where][PrimerApellido]=' + passwordUse
r, {json: true}, (error, response) => {
      let isRegistered = '';
      let id = '';
      if (response.body[0].Nombre == '') {
        registered = [isRegistered, id];
        segundoApellido = ' ';
      } else {
        isRegistered = response.body[0].UsuarioMoodle;
        id = response.body[0].id;
        registered = [isRegistered, id];
      }
      obs.next(registered);
    });
  });
  return registeredObservable;
}
```

Comprobamos en la base de datos si el alumno está o no registrado con Moodle y también si el alumno está registrado en Classpip.

La página de login creada en *ejs* es la siguiente:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <%if (data.classpipRegister == false) {%>
    <body onload="alerta2()"></body>
  <% }%>
  <body onload="alerta()"></body>
  <title>Login Page</title>
  <style>
  .login-form {
    width: 300px;
    margin: 0 auto;
```

```
    font-family: Tahoma, Geneva, sans-serif;
  }
  .login-form h1 {
    text-align: center;
    color: #4d4d4d;
    font-size: 24px;
    padding: 20px 0 20px 0;
  }
  .login-form input[type="password"],
  .login-form input[type="text"] {
    width: 100%;
    padding: 15px;
    border: 1px solid #dddddd;
    margin-bottom: 15px;
    box-sizing: border-box;
  }
  .login-form input[type="submit"] {
    width: 100%;
    padding: 15px;
    background-color: #535b63;
    border: 0;
    box-sizing: border-box;
    cursor: pointer;
    font-weight: bold;
    color: #ffffff;
  }
  .footer {
    width: 70;
    margin: 5% auto;
  }
  </style>
</head>
<body>
  <%if (data.registered[0] == '') {%>
  <div class="login-form">
    <h1>Login</h1>
    <form action="/loginUser" method="POST">
      <input type="text" id ="username" name="username" value="<
%=data.username%>" required>
      <input type="password" id ="password" name="password" valu
e="<%=data.password%>" required>
      <input type="submit">
```

```
    </form>
  </div>
  <% }%>
  <%if (data.registered[0] != '') {%>
    <META HTTP-
EQUIV="REFRESH" CONTENT="1;URL=http://localhost:3000/loginUser/<
%=data.registered[1]%>">
  <% }%>
</body>

<script type="text/javascript">
  function alerta() {
    alert('\nSi es tu primera vez usando la herramienta ClassPip
, tendrás que iniciar sesión introduciendo tu usuario y contrase
ña\n\nSi todavía no estás registrado, crearás una cuenta asociad
a a tu usuario de Moodle cuando le des al botón de enviar\n\nPor
 defecto se utilizan los datos de Moodle');
  }
  function alerta2() {
    alert('No estás registrado en Classpip. Registrate primero p
ara poder acceder a tu cuenta');
  }
</script>

</html>
```

Lo destacable de este fichero es que tenemos un formulario en el que se introduce el nombre de usuario y la contraseña y que en caso de que el usuario esté registrado, seremos redirigidos directamente a la página con el listado de juegos del alumno.

Si el usuario no está registrado, se mostrará una alerta indicándolo.

6. Guía para implementar la lista de juegos de un alumno

Lo primero de todo es crear la ruta para manejar la redirección a la página que contendrá el listado de juegos (una vez iniciada la sesión):

```
app.use('/loginUser', function(req, resp, next) {
    let idRegisteredUser = req.url.split('/');
    if (idRegisteredUser[1] != '' && nameUser != '') {
      userId = idRegisteredUser[1];
      getUser(nameUser, passUser).subscribe(res => {
     getListaJuegos(idRegisteredUser[1]).subscribe(response => {
          Juegos = response;
          resp.render('..\\..\\views\\games.ejs', {Nombre: nameU
ser, table: Juegos, Avatar: miAvatar});
        });
      });
    } else if (nameUser == '') {
      let userData = {username: nameUser, password: passUser, re
gistered: miRegisteredUser, classpipRegister: false};
      resp.render('..\\..\\views\\login.ejs', {data: userData});
    } else {
      nombreUser = req.body.username;
getUser(req.body.username, req.body.password).subscribe(res => {
        userId = res;
        alumno = JSON.stringify({Nombre: nameUser, PrimerApellid
o: passUser, SegundoApellido: segundoApellido,
          ImagenPerfil: miAvatarRoot, UsuarioMoodle: idUserMoodl
e, id: userId, profesorId: profesorId});
        updateUser(alumno, userId);
        getListaJuegos(userId).subscribe(response => {
          console.log('estamos en el login user 3: ' + JSON.stri
ngify(response));
          Juegos = response;
          resp.render('..\\..\\views\\games.ejs', {Nombre: nombr
eUser, table: Juegos, Avatar: miAvatar});
        });
      });
    }
  });
 });
```

Como podemos ver, primero se comprueba si es un inicio de sesión para un usuario ya registrado o que se registra por primera vez. Este parámetro lo sacamos de la *url* de la petición:

```
let idRegisteredUser = req.url.split('/');
 if (idRegisteredUser[1] != '') {
```

Si el usuario ya está registrado, llamamos a dos funciones para obtener los datos del alumno y su listado de juegos:

```
getUser(nameUser, passUser).subscribe(res => {
     getListaJuegos(idRegisteredUser[1]).subscribe(response => {
         Juegos = response;
```

Se puede apreciar que en la primera función le pasamos el nombre de usuario y su contraseña de variables globales. Estas variables se crean nada más se conecta Moodle con Classpip mediante LTIJS:

```
  app.get('/begin', lti.app, async (req, res) => {
    // Render with ejs the login page
    registeredUser(nameUser, passUser).subscribe(registration =>
 {
      console.log('ESTAMOS EN REGISTEREDUSER RESPONSE');
      miRegisteredUser = registration;
      let userData = {username: nameUser, password: passUser, re
gistered: miRegisteredUser, classpipRegister: true};
      res.render('login.ejs', {data: userData});
    });
  });
```

En esta parte, llamamos a la función *registeredUser(nameUser, passUser)* para saber si el usuario ya se ha registrado.

Esta función consulta en la base de datos si el campo *UsuarioMoodle* está activo (true). La función la tenemos explicada en la anterior sección.

En caso de que no esté registrado, sacamos el nombre de usuario de la *url* de la petición y llamamos también a las dos funciones anteriores:

```
 nombreUser = req.body.username;
getUser(req.body.username, req.body.password).subscribe(res => {
        userId = res;
        alumno = JSON.stringify({Nombre: nameUser, PrimerApellid
o: passUser, SegundoApellido: segundoApellido,
          ImagenPerfil: miAvatarRoot, UsuarioMoodle: true, id: u
serId, profesorId: profesorId});
```

```
        updateUser(alumno, userId);
        getListaJuegos(userId).subscribe(response => {
          console.log('estamos en el login user 3: ' + JSON.stri
ngify(response));
          Juegos = response;
          resp.render('..\\..\\views\\games.ejs', {Nombre: nombr
eUser, table: Juegos, Avatar: miAvatar});
```

A diferencia del anterior inicio de sesión, el nombre y la contraseña del alumno se sacan de la *url* de la petición. El *id* se obtiene como respuesta a la función de *getUser(nombreUsuario, contraseñaUsuario)*.

Aprovechamos para crear un modelo de Alumno con los datos del alumno en cuestión. Estos datos los sacamos de diferentes variables globales que se rellenan a medida que los datos se extraen del inicio de sesión de Moodle.

Como se puede ver, antes de conseguir la lista de juegos, realizamos la actualización del alumno para modificar el campo de MoodleUser:

```
function updateUser(miAlumno, idUser) {
  let url = server  + '/api/Alumnos/' + idUser;
  request({url: url, method: 'PUT', json: JSON.parse(miAlumno)},
 function(error2, request2, body) {
  });
}
```

Con esto, cargamos el nuevo alumno en la base de datos.

En cuanto a la función de getUser:

```
function getUser(Name, Surname) {
  const userIdObservable = new Observable(obs => {
    let userId = '';
    request.get(server + '/api/Alumnos?filter[where][Nombre]=' +
      Name + '&filter[where][PrimerApellido]=' + Surname, {json:
 true}, (error, response) => {
        userId = response.body[0].id;
        segundoApellido = response.body[0].SegundoApellido;
        profesorId = response.body[0].profesorId;
        miAvatar = server + '/api/imagenes/imagenAlumno/download
/' + response.body[0].ImagenPerfil;
        miAvatarRoot = response.body[0].ImagenPerfil;
        request.get(miAvatar, function(error, response, body) {
          console.error('response.statusCode:', response.statusC
ode);
```

```
          if (response.statusCode == 404) {
            miAvatar = server + '/api/imagenes/imagenAlumno/down
load/UsuarioAlumno.jpg';
          }
        });
        obs.next(userId);
      });
  });
  return userIdObservable;
}
```

Realizamos la consulta en la base de datos para extraer los datos del alumno a partir del nombre de usuario y la contraseña. En estos momentos corresponden al Nombre y PrimerApellido del alumno. También cargamos la imagen de su avatar y en caso de no tener ninguna, se le asigna una por defecto.

La función para cargar la lista de juegos es la más extensa de todas, ya que implica a muchas otras funciones tal y como vemos a continuación:

```
function getListaJuegos(idUser) {
  const listaObservables = new Observable(obs => {
    let listaJuegos = [];
    let bPuntos = [];
    let cPuntos = [];
    let dPuntos = [];
    let ePuntos = [];
    let fPuntos = [];
    // JUEGO DE PUNTOS
    getAlumnoJuegoPuntos(idUser).subscribe(alumnojuegodepuntos =
> {
      bPuntos = alumnojuegodepuntos[0];
      let size = alumnojuegodepuntos[1];
      let uri = alumnojuegodepuntos[2];
      console.log('MY URI 2222 IN %s is: ' + uri);
      getJuegoDePuntos(uri).subscribe(juegodepuntos => {
        let body2 = juegodepuntos[0];
        let size2 = juegodepuntos[1];
        if (size2 <= 0) {
          for (let j = 0; j < size2; j ++) {
            listaJuegos.push({NombreJuego: '', Puntos: [bPuntos[
j].PuntosTotalesAlumno],
              Modo: '', JuegoActivo: 'null', id: '', Tipo: ''});
```

```
            console.log('MI PUNTOS SON: ' + bPuntos[j].PuntosTot
alesAlumno);
            console.log('2 MIS JUEGOS ACTIVOS SON: ' + listaJueg
os);
          }
        } else {
          for (let j = 0; j < size; j ++) {
            listaJuegos.push({NombreJuego: [body2[j].NombreJuego
], Puntos: [bPuntos[j].PuntosTotalesAlumno],
              Modo: [body2[j].Modo], JuegoActivo: [body2[j].Jueg
oActivo], id: [body2[j].id], Tipo: [body2[j].Tipo],
              idAlumno: idUser});
            console.log('MI PUNTOS SON: ' + bPuntos[j].PuntosTot
alesAlumno);
            console.log('2 MIS JUEGOS ACTIVOS SON: ' + listaJueg
os);
          }
        }
        console.log('MIS JUEGOS ACTIVOS (PUNTOS) SON: ' + JSON.s
tringify(listaJuegos));
        // JUEGO COMPETICION LIGA
        getAlumnoJuegoCompeticionLiga(idUser).subscribe(alumnoju
egocompeticionliga => {
          cPuntos = alumnojuegocompeticionliga[0];
          let size3 = alumnojuegocompeticionliga[1];
          let uri2 =  alumnojuegocompeticionliga[2];
          getJuegoCompeticionLiga(uri2).subscribe(juegocompetici
onliga => {
            let body4 = juegocompeticionliga[0];
            let size4 = juegocompeticionliga[1];
            if (size4 > 0) {
              for (let l = 0; l < size4; l++) {
                listaJuegos.push({NombreJuego: [body4[l].NombreJ
uego], Puntos: [cPuntos[l].PuntosTotalesAlumno],
                  Modo: [body4[l].Modo], JuegoActivo: [body4[l].
JuegoActivo], id: [body4[l].id], Tipo: [body4[l].Tipo],
                  Jornadas: [body4[l].NumeroTotalJornadas]});
                console.log('MI PUNTOS SON: ' + cPuntos[l].Punto
sTotalesAlumno);
                console.log('3 MIS JUEGOS ACTIVOS SON: ' + lista
Juegos);
              }
```

```
            } else {
              for (let l = 0; l < size3; l++) {
                listaJuegos.push({NombreJuego: '', Puntos: cPunt
os[l].PuntosTotalesAlumno,
                  Modo: '', JuegoActivo: 'null', id: '', Tipo: '
',
                  Jornadas: ''});
                console.log('MI PUNTOS SON: ' + cPuntos[l].Punto
sTotalesAlumno);
                console.log('3 MIS JUEGOS ACTIVOS SON: ' + lista
Juegos);
              }
            }
            console.log('MIS JUEGOS ACTIVOS (competicion liga) S
ON: ' + JSON.stringify(listaJuegos));
            // JUEGO COMPETICION LIGA FORMULA UNO
            getAlumnoJuegoCompeticionFUno(idUser).subscribe(alum
nojuegocompeticionfu => {
              dPuntos = alumnojuegocompeticionfu[0];
              let size5 = alumnojuegocompeticionfu[1];
              let uri3 = alumnojuegocompeticionfu[2];
              getJuegoCompeticionFU(uri3).subscribe(juegocompeti
cionfu => {
                let body6 = juegocompeticionfu[0];
                let size6 = juegocompeticionfu[1];
                if (size6 > 0) {
                  for (let n = 0; n < size6; n++) {
                    listaJuegos.push({NombreJuego: [body6[n].Nom
breJuego], Puntos: dPuntos[n].PuntosTotalesAlumno,
                      Modo: body6[n].Modo, JuegoActivo: [body6[n
].JuegoActivo], id: [body6[n].id], Tipo: [body6[n].Tipo],
                      Jornadas: [body6[n].NumeroTotalJornadas],
NumeroParticipantesPuntuan: [body6[n].NumeroParticipantesPuntuan
],
                      PuntosArray: [body6[n].Puntos]});
                    console.log('MI PUNTOS SON: ' + dPuntos[n].P
untosTotalesAlumno);
                    console.log('4 MIS JUEGOS ACTIVOS SON: ' + J
SON.stringify(listaJuegos));
                  }
                } else {
                  for (let n = 0; n < size5; n++) {
```

```
                    listaJuegos.push({NombreJuego: '', Puntos: d
Puntos[n].PuntosTotalesAlumno,
                      Modo: '', JuegoActivo: 'null', id: '', Tip
o: '',
                      Jornadas: '', NumeroParticipantesPuntuan:
'',
                      PuntosArray: ''});
                    console.log('MI PUNTOS SON: ' + dPuntos[n].P
untosTotalesAlumno);
                    console.log('4 MIS JUEGOS ACTIVOS SON: ' + J
SON.stringify(listaJuegos));
                  }
                }
                console.log('MIS JUEGOS ACTIVOS (competicion for
mula formula uno) SON: ' + JSON.stringify(listaJuegos));
                // JUEGO CUESTIONARIO
                getAlumnoJuegoCuestionario(idUser).subscribe(alu
mnojuegocuestionario => {
                  ePuntos = alumnojuegocuestionario[0];
                  let size7 = alumnojuegocuestionario[0];
                  let uri4 = alumnojuegocuestionario[2];
                  getJuegoCuestionario(uri4).subscribe(juegocues
tionario => {
                    let body8 = juegocuestionario[0];
                    let size8 = juegocuestionario[1];
                    if (size8 > 0) {
                      for (let o = 0; o < size8; o++) {
                        listaJuegos.push({NombreJuego: [body8[o]
.NombreJuego], Puntos: ePuntos[o].Nota,
                          cuestionarioId: body8[o].cuestionarioI
d, JuegoActivo: [body8[o].JuegoActivo], id: [body8[o].id], Juego
Terminado: [body8[o].JuegoTerminado],
                          Presentacion: [body8[o].Presentacion],
 PuntuacionCorrecta: [body8[o].PuntuacionCorrecta],
                          PuntuacionIncorrecta: [body8[o].Puntua
cionIncorrecta], Tipo: ['Juego De Cuestionario'], Modo: ['Indivi
dual'],
                          grupoId: body8[o].grupoId, juegoDeCues
tionarioId: body8[o].id, idCuestionarioAlumnoJuego: ePuntos[o].i
d, idAlumno: idUser});
                        console.log('MI PUNTOS SON: ' + ePuntos[
o].Nota);
```

```
                        console.log('4 MIS JUEGOS ACTIVOS SON: '
 + JSON.stringify(listaJuegos));
                      }
                    } else {
                      for (let o = 0; o < size7; o++) {
                        listaJuegos.push({NombreJuego: '', Punto
s: ePuntos[o].Nota,
                          cuestionarioId: '', JuegoActivo: 'null
', id: '', JuegoTerminado: '',
                          Presentacion: '', PuntuacionCorrecta:
'',
                          PuntuacionIncorrecta: '', Tipo: ['Jueg
o De Cuestionario'], Modo: ['Individual'],
                          grupoId: '', juegoDeCuestionarioId: ''
, idCuestionarioAlumnoJuego: ePuntos[o].id, idAlumno: idUser});
                        console.log('MI PUNTOS SON: ' + ePuntos[
o].Nota);
                        console.log('4 MIS JUEGOS ACTIVOS SON: '
 + JSON.stringify(listaJuegos));
                      }
                    }
                    console.log('MIS JUEGOS ACTIVOS (cuestionari
o) SON: ' + JSON.stringify(listaJuegos));
                    getAlumnoJuegoColeccion(idUser).subscribe(al
umnojuegocoleccion => {
                      fPuntos = alumnojuegocoleccion[0];
                      let size9 = alumnojuegocoleccion[1];
                      let uri5 = alumnojuegocoleccion[2];
                      getJuegoColeccion(uri5).subscribe(juegocol
eccion => {
                        let body9 = juegocoleccion[0];
                        let size10 = juegocoleccion[1];
                        if (size9 > 0) {
                          for (let o = 0; o < size9; o++) {
                            listaJuegos.push({NombreJuego: [body
9[o].NombreJuego], coleccionId: body9[o].coleccionId,
                              JuegoActivo: [body9[o].JuegoActivo
], id: [body9[o].id], Tipo: [body9[o].Tipo], Modo: ['Individual'
],
                              grupoId: body9[o].grupoId, juegoDe
ColeccionId: body9[o].id, idColeccionAlumnoJuego: fPuntos[o].id,
 idAlumno: idUser});
```

```
                            console.log('5 MIS JUEGOS ACTIVOS SO
N: ' + JSON.stringify(listaJuegos));
                          }
                        } else {
                          for (let o = 0; o < size10; o++) {
                            listaJuegos.push({NombreJuego: '', c
oleccionId: '', JuegoActivo: 'null',
                              id: '', Tipo: [body9[o].Tipo], Mod
o: ['Individual'],
                              grupoId: '', juegoDeColeccionId: '
', idColeccionAlumnoJuego: fPuntos[o].id, idAlumno: idUser});
                            console.log('5 MIS JUEGOS ACTIVOS SO
N: ' + JSON.stringify(listaJuegos));
                          }
                        }
                        console.log('MIS JUEGOS ACTIVOS (colecci
on) SON: ' + JSON.stringify(listaJuegos));
                        getAlumnoJuegoAvatar(idUser).subscribe(a
lumnojuegosavatar => {
                          let mydata = alumnojuegosavatar[0];
                          let size10 = alumnojuegosavatar[1];
                          let uri6 = alumnojuegosavatar[2];
                          getJuegosAvatar(uri6).subscribe(juegos
avatar => {
                            let body10 = juegosavatar[0];
                            let size11 = juegosavatar[1];
                            if (size10 > 0) {
                              for (let p = 0; p < size10; p++) {
                                listaJuegos.push({NombreJuego: [
body10[p].NombreJuego],
                                  JuegoActivo: [body10[p].JuegoA
ctivo], juegoAvatarid: [body10[p].id], Tipo: [body10[p].Tipo], M
odo: [body10[p].Modo],
                                  grupoId: body10[p].grupoId, Cr
iteriosPrivilegioComplemento1: body10[p].CriteriosPrivilegioComp
lemento1,
                                  CriteriosPrivilegioComplemento
2: [body10[p].CriteriosPrivilegioComplemento2],
                                  CriteriosPrivilegioComplemento
3: [body10[p].CriteriosPrivilegioComplemento3],
                                  CriteriosPrivilegioComplemento
4: [body10[p].CriteriosPrivilegioComplemento4],
```

```
                                    CriteriosPrivilegioVoz: [body1
0[p].CriteriosPrivilegioVoz],
                                    CriteriosPrivilegioVerTodos: [
body10[p].CriteriosPrivilegioVerTodos]});
                                  console.log('6 MIS JUEGOS ACTIVO
S SON: ' + JSON.stringify(listaJuegos));
                                }
                              } else {
                                for (let q = 0; q < size11; q++) {
                                  listaJuegos.push({NombreJuego: '
', coleccionId: '', JuegoActivo: 'null',
                                    id: '', Tipo: [body10[q].Tipo]
, Modo: [body10[q].Modo]});
                                  console.log('6 2 MIS JUEGOS ACTI
VOS SON: ' + JSON.stringify(listaJuegos));
                                }
                              }
                              console.log('MIS JUEGOS ACTIVOS (AVA
TAR) SON: ' + JSON.stringify(listaJuegos));
                              obs.next(listaJuegos);
                            });
                          });
                        });
                      });
                    });
                  });
                });
              });
            });
          });
        });
      });
    });
  });
  return listaObservables;
}
```

Se puede ver que hay diversas funciones que nos permiten cargar los datos necesarios para cada juego y además, nos facilitan las URL para realizar las siguientes peticiones:

Función getAlumnoJuegoPuntos():

```
function getAlumnoJuegoPuntos(idAlumno) {
```

```
  const alumnoJuegoPuntosbservable = new Observable(obs => {
    let alumnoJuegoDePuntos = [];
    request.get(server  + '/api/AlumnoJuegosDePuntos?filter[wher
e][alumnoId]=' +
    idAlumno, {json: true}, (error, response, body) => {
      alumnoJuegoDePuntos.push(body);
      // Recogemos valores del body
      let uri = '';
      let size = Object.keys(body).length;
      alumnoJuegoDePuntos.push(size);
      console.log('1111111111 L body: ', size); // saber la long
itud del body (cuantos elementos tenemos)
      console.log('1111111 body: ', body);
      let filterone = '';
      if (size > 0) {
        filterone = body[0].juegoDePuntosId;
      }
      uri = server  + '/api/JuegosDePuntos?filter[where][or][0][
id]=' + filterone;
      if (body.length > 1) {
        for (let i = 1; i < body.length; i++) {
          uri += '&filter[where][or]' + '[' + i + ']' + '[id]='
+ body[i].juegoDePuntosId;
          console.log('MY URI IN %s is: ' + uri, i);
        }
      }
      alumnoJuegoDePuntos.push(uri);
      obs.next(alumnoJuegoDePuntos);
    });
  });
  return alumnoJuegoPuntosbservable;
}
```

A partir de la uri de la función anterior, podemos proceder a la siguiente.

Función getJuegoDePuntos():

```
function getJuegoDePuntos(myuri, type) {
  const juegoPuntosbservable = new Observable(obs => {
    let juegoDePuntos = [];
    request.get(myuri, {json: true}, (error, response, body2) =>
 {
      if (type == 1) {
```

```
        juegoDePuntos = body2;
      } else {
        juegoDePuntos.push(body2);
        console.log('2222222 body: ', body2);
        // Recogemos valores del body
        let size2 = body2.length;
        juegoDePuntos.push(size2);
        console.log('22222222 L body: ', size2); // saber la lon
gitud del body (cuantos elementos tenemos)
      }
      obs.next(juegoDePuntos);
    });
  });
  return juegoPuntosbservable;
}
```

Estos datos se introducen, mediante un recorrido por cada lista, dentro de una lista donde tendremos todos los datos de los juegos:

```
            let body4 = juegocompeticionliga[0];
            let size4 = juegocompeticionliga[1];
            if (size4 > 0) {
              for (let l = 0; l < size4; l++) {
                listaJuegos.push({NombreJuego: [body4[l].NombreJ
uego], Puntos: [cPuntos[l].PuntosTotalesAlumno],
                  Modo: [body4[l].Modo], JuegoActivo: [body4[l].
JuegoActivo], id: [body4[l].id], Tipo: [body4[l].Tipo],
                  Jornadas: [body4[l].NumeroTotalJornadas]});
                console.log('MI PUNTOS SON: ' + cPuntos[l].Punto
sTotalesAlumno);
                console.log('3 MIS JUEGOS ACTIVOS SON: ' + lista
Juegos);
              }
            } else {
              for (let l = 0; l < size3; l++) {
                listaJuegos.push({NombreJuego: '', Puntos: cPunt
os[l].PuntosTotalesAlumno,
                  Modo: '', JuegoActivo: 'null', id: '', Tipo: '
',
                  Jornadas: ''});
                console.log('MI PUNTOS SON: ' + cPuntos[l].Punto
sTotalesAlumno);
```

```
                console.log('3 MIS JUEGOS ACTIVOS SON: ' + lista
Juegos);
              }
            }
```

Función getAlumnoJuegoCompeticionLiga:

```
function getAlumnoJuegoCompeticionLiga(idUser) {
  const alumnoJuegoCompeticionLigabservable = new Observable(obs
 => {
    let alumnoJuegoCompeticionLiga = [];
    request.get(server  + '/api/AlumnosJuegoDeCompeticionLiga?fi
lter[where][AlumnoId]=' +
      idUser, {json: true}, (error, response, body3) => {
        alumnoJuegoCompeticionLiga.push(body3);
        // Recogemos valores del body
        let uri2 = '';
        let size3 = body3.length;
        alumnoJuegoCompeticionLiga.push(size3);
        console.log('3333333 L body: ', size3); // saber la long
itud del body (cuantos elementos tenemos)
        console.log('3333333 body: ', body3);
        let filterone2 = '';
        if (size3 > 0) {
          filterone2 = body3[0].JuegoDeCompeticionLigaId;
        }
        uri2 = server  + '/api/JuegosDeCompeticionLiga?filter[wh
ere][or][0][id]=' + filterone2;
        if (body3.length > 1) {
          for (let k = 1; k < size3; k++) {
            uri2 += '&filter[where][or]' + '[' + k + ']' + '[id]
=' + body3[k].JuegoDeCompeticionLigaId;
            console.log('MY URI IN %s is: ' + uri2, k);
          }
        }
        alumnoJuegoCompeticionLiga.push(uri2);
        obs.next(alumnoJuegoCompeticionLiga);
      });
  });
  return alumnoJuegoCompeticionLigabservable;
}
```

Función getJuegoCompeticionLiga:

```
function getJuegoCompeticionLiga(myuri) {
  const juegoJuegoCompeticionLigabservable = new Observable(obs
=> {
    let juegoCompeticionLiga = [];
    request.get(myuri, {json: true}, (error, response, body4) =>
 {
      console.log('4444444 body: ', body4);
      juegoCompeticionLiga.push(body4);
      // Recogemos valores del body
      let size4 = body4.length;
      juegoCompeticionLiga.push(size4);
      console.log('4444444 L body: ', size4); // saber la longit
ud del body (cuantos elementos tenemos)
      console.log('444444 body: ', body4);
      obs.next(juegoCompeticionLiga);
    });
  });
  return juegoJuegoCompeticionLigabservable;
}
```

Igual que antes, lo guardamos en la lista global:

```
            let body4 = juegocompeticionliga[0];
            let size4 = juegocompeticionliga[1];
            if (size4 > 0) {
              for (let l = 0; l < size4; l++) {
                listaJuegos.push({NombreJuego: [body4[l].NombreJ
uego], Puntos: [cPuntos[l].PuntosTotalesAlumno],
                  Modo: [body4[l].Modo], JuegoActivo: [body4[l].
JuegoActivo], id: [body4[l].id], Tipo: [body4[l].Tipo],
                  Jornadas: [body4[l].NumeroTotalJornadas]});
                console.log('MI PUNTOS SON: ' + cPuntos[l].Punto
sTotalesAlumno);
                console.log('3 MIS JUEGOS ACTIVOS SON: ' + lista
Juegos);
              }
            } else {
              for (let l = 0; l < size3; l++) {
                listaJuegos.push({NombreJuego: '', Puntos: cPunt
os[l].PuntosTotalesAlumno,
                  Modo: '', JuegoActivo: 'null', id: '', Tipo: '
',
```

```
                  Jornadas: ''});
                console.log('MI PUNTOS SON: ' + cPuntos[l].Punto
sTotalesAlumno);
                console.log('3 MIS JUEGOS ACTIVOS SON: ' + lista
Juegos);
              }
            }
```

El resto de juegos sigue el mismo patrón, por lo que dejaré las funciones utilizadas.

Función getAlumnoJuegoCompeticionFUno:

```
function getAlumnoJuegoCompeticionFUno(idUser) {
  const alumnoJuegoCompeticionFUnobservable = new Observable(obs
 => {
    let alumnoJuegoCompeticionFUno = [];
    request.get(server  + '/api/AlumnosJuegoDeCompeticionFormula
Uno?filter[where][AlumnoId]=' +
      idUser, {json: true}, (error, response, body5) => {
        alumnoJuegoCompeticionFUno.push(body5);
        // Recogemos valores del body
        let uri3 = '';
        let size4 = body5.length;
        alumnoJuegoCompeticionFUno.push(size4);
        console.log('5555555 L body: ', size4); // saber la long
itud del body (cuantos elementos tenemos)
        console.log('5555555 body: ', body5);
        let filterone3 = '';
        if (size4 > 0) {
          filterone3 = body5[0].JuegoDeCompeticionFormulaUnoId;
        }
        uri3 = server  + '/api/JuegosDecompeticionFormulaUno?fil
ter[where][or][0][id]=' + filterone3;
        if (body5.length > 1) {
          for (let m = 1; m < size4; m++) {
            uri3 += '&filter[where][or]' + '[' + m + ']' + '[id]
=' + body5[m].JuegoDeCompeticionFormulaUnoId;
            console.log('MY URI IN %s is: ' + uri3, m);
          }
        }
        alumnoJuegoCompeticionFUno.push(uri3);
        obs.next(alumnoJuegoCompeticionFUno);
```

```
      });
  });
  return alumnoJuegoCompeticionFUnobservable;
}
```

Función getJuegoCompeticionFU:

```
function getJuegoCompeticionFU(myuri) {
  const juegoJuegoCompeticionFUObservable = new Observable(obs =
> {
    let juegoCompeticionFU = [];
    request.get(myuri, {json: true}, (error, response, body6) =>
 {
      juegoCompeticionFU.push(body6);
      console.log('6666666 body: ', body6);
      // Recogemos valores del body
      let size6 = body6.length;
      juegoCompeticionFU.push(size6);
      console.log('6666666 L body: ', size6); // saber la longit
ud del body (cuantos elementos tenemos)
      console.log('666666 body: ', body6);
      obs.next(juegoCompeticionFU);
    });
  });
  return juegoJuegoCompeticionFUObservable;
}
```

Función getAlumnoJuegoCuestionario:

```
function getAlumnoJuegoCuestionario(idUser) {
  const alumnoJuegoCuestionarioObservable = new Observable(obs =
> {
    let alumnoJuegoCuestionario = [];
    request.get(server  + '/api/AlumnosJuegoDeCuestionario?filte
r[where][alumnoId]=' +
    idUser, {json: true}, (error, response, body7) => {
      // Recogemos valores del body
      alumnoJuegoCuestionario.push(body7);
      let uri4 = '';
      let size6 = body7.length;
      alumnoJuegoCuestionario.push(size6);
      console.log('77777777 L body: ', size6); // saber la longi
tud del body (cuantos elementos tenemos)
```

```
      console.log('777777777 body: ', body7);
      let filterone4 = '';
      if (size6 > 0) {
        filterone4 =  body7[0].juegoDeCuestionarioId;
      }
      uri4 = server  + '/api/JuegosDeCuestionario?filter[where][
or][0][id]=' + filterone4;
      console.log('111111777777 MY URI IN %s is: ' + uri4);
      if (body7.length > 1) {
        for (let o = 1; o < size6; o ++) {
          uri4 += '&filter[where][or]' + '[' + o + ']' + '[id]='
 + body7[o].juegoDeCuestionarioId;
          console.log('777777777 MY URI IN %s is: ' + uri4, o);
        }
      }
      alumnoJuegoCuestionario.push(uri4);
      obs.next(alumnoJuegoCuestionario);
    });
  });
  return alumnoJuegoCuestionarioObservable;
}
```

Función getJuegoCuestionario:

```
function getJuegoCuestionario(myuri) {
  const juegoJuegoCuestionarioUObservable = new Observable(obs =
> {
    let juegoCuestionario = [];
    request.get(myuri, {json: true}, (error, response, body10) =
> {
      juegoCuestionario.push(body10);
      console.log('88888888 body: ', body10);
      // Recogemos valores del body
      let size8 = body10.length;
      juegoCuestionario.push(size8);
      console.log('888888888 L body: ', size8); // saber la long
itud del body (cuantos elementos tenemos)
      console.log('888888888 body: ', body10);
      obs.next(juegoCuestionario);
    });
  });
  return juegoJuegoCuestionarioUObservable;
```

```
}
```

Función getAlumnoJuegoColeccion:

```
function getAlumnoJuegoColeccion(idUser) {
  const alumnoJuegoColeccionObservable = new Observable(obs => {
    let alumnoJuegoColeccion = [];
    request.get(server  + '/api/AlumnosJuegoDeColeccion?filter[where][alumnoId]=' +
    idUser, {json: true}, (error, response, body9) => {
      // Recogemos valores del body
      alumnoJuegoColeccion.push(body9);
      let uri5 = '';
      let size9 = body9.length;
      alumnoJuegoColeccion.push(size9);
      console.log('8888888 L body: ', size9); // saber la longitud del body (cuantos elementos tenemos)
      console.log('8888888 body: ', body9);
      let filterone4 = '';
      if (size9 > 0) {
        filterone4 =  body9[0].juegoDeColeccionId;
      }
      uri5 = server  + '/api/JuegosDeColeccion?filter[where][or][0][id]=' + filterone4;
      console.log('1118888888 MY URI IN %s is: ' + uri5);
      if (body9.length > 1) {
        for (let o = 1; o < size9; o ++) {
          uri5 += '&filter[where][or]' + '[' + o + ']' + '[id]=' + body9[o].juegoDeColeccionId;
          console.log('8888888 MY URI IN %s is: ' + uri5, o);
        }
      }
      alumnoJuegoColeccion.push(uri5);
      obs.next(alumnoJuegoColeccion);
    });
  });
  return alumnoJuegoColeccionObservable;
}
```

Función getJuegoColeccion:

```
function getJuegoColeccion(myuri) {
```

```
  const juegoJuegoColeccionsUObservable = new Observable(obs =>
{
    let juegoColeccion = [];
    request.get(myuri, {json: true}, (error, response, body8) =>
 {
      juegoColeccion.push(body8);
      console.log('88888888 body: ', body8);
      // Recogemos valores del body
      let size10 = body8.length;
      juegoColeccion.push(size10);
      console.log('888888888 L body: ', size10); // saber la lon
gitud del body (cuantos elementos tenemos)
      obs.next(juegoColeccion);
    });
  });
  return juegoJuegoColeccionsUObservable;
}
```

Función getAlumnoJuegoAvatar:

```
function getAlumnoJuegoAvatar(idAlumno) {
  const juegoAvatarObservable = new Observable(obs => {
    let misJuegosDeAvatar = [];
    let myuri = server + '/api/alumnosJuegoAvatar?filter[where][
alumnoId]=' + idAlumno;
    let uri3 = '';
    request.get(myuri, {json: true}, (error, response, body8) =>
 {
      misJuegosDeAvatar.push(body8);
      misJuegosDeAvatar.push(body8.length);
      console.log('15 misJuegosDeAvatar: ', misJuegosDeAvatar);
      console.log('15 body8: ', body8.length);
      let filterone3 = '';
      if (body8.length > 0) {
        filterone3 = body8[0].juegoDeAvatarId;
      }
      uri3 = server  + '/api/JuegosDeAvatar?filter[where][or][0]
[id]=' + filterone3;
      if (body8.length > 1) {
        for (let m = 1; m < body8.length; m++) {
          uri3 += '&filter[where][or]' + '[' + m + ']' + '[id]='
 + body8[m].juegoDeAvatarId;
```

```
          console.log('MY URI getAlumnoJuegoAvatar IN %s is: ' +
 uri3, m);
        }
      }
      misJuegosDeAvatar.push(uri3);
      obs.next(misJuegosDeAvatar);
    });
  });
  return juegoAvatarObservable;
}
```

Función getJuegosAvatar:

```
function getJuegosAvatar(myuri) {
  const juegoAvatarObservable = new Observable(obs => {
    let juegoAvatar = [];
    request.get(myuri, {json: true}, (error, response, body6) =>
 {
      juegoAvatar.push(body6);
      console.log('16 body: ', body6);
      // Recogemos valores del body
      let size6 = body6.length;
      juegoAvatar.push(size6);
      console.log('16 L body: ', size6); // saber la longitud de
l body (cuantos elementos tenemos)
      obs.next(juegoAvatar);
    });
  });
  return juegoAvatarObservable;
}
```

Finalmente, se acaba renderizando la página con los juegos del alumno:

```
resp.render('..\\..\\views\\games.ejs', {Nombre: nombreUser, tab
le: Juegos, Avatar: miAvatar});
```

La variable miAvatar también se obtiene de la función getUser(*nombreUsuario, contraseñaUsuario*).

En cuanto a la página ejs:

```
<body class="text-center">
```

```
<div class="cover-container d-flex w-100 h-100 p-3 mx-auto flex-column">
<html>
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
<style>
  td, th {
  border: 1px solid #dddddd;
  text-align: left;
  padding: 8px;
  font-family: Tahoma, Geneva, sans-serif;
  }

  tr:nth-child(even) {
  background-color: #d8d9d9;
  font-family: Tahoma, Geneva, sans-serif;
  }

  .button {
    width: 100%;
    padding: 5px;
    background-color: #0f2d4b;
    border: 0;
    box-sizing: border-box;
    cursor: pointer;
    font-weight: bold;
    color: #ffffff;
    font-family: Tahoma, Geneva, sans-serif;
  }

  .buttonb {
    width: 8%;
    padding: 5px;
    background-color: #0f2d4b;
    border: 0;
    box-sizing: border-box;
    cursor: pointer;
    font-weight: bold;
    color: #ffffff;
    font-family: Tahoma, Geneva, sans-serif;
  }
```

```css
.login-form:hover input[type="submit"]{
  background-color: #78c57a;
  color: white;
  font-family: Tahoma, Geneva, sans-serif;
}

.buttonRankPuntos:hover {
  background-color: #057b6dbe;
  color: white;
  font-family: Tahoma, Geneva, sans-serif;
}

.buttonCuestionario:hover {
  background-color: #49c112b1;
  color: white;
  font-family: Tahoma, Geneva, sans-serif;
}

.buttonColeccion:hover {
  background-color: #dd6313b1;
  color: white;
  font-family: Tahoma, Geneva, sans-serif;
}

.buttonAvatar:hover {
  background-color: #1a90d9b1;
  color: white;
  font-family: Tahoma, Geneva, sans-serif;
}

.buttonBack:hover {
  background-color: #952807;
  color: white;
  font-family: Tahoma, Geneva, sans-serif;
}

.login-form input[type="submit"] {
  width: 100%;
  padding: 5px;
  background-color: #002d5b;
  border: 0;
  box-sizing: border-box;
```

```
    cursor: pointer;
    font-weight: bold;
    color: #ffffff;
    font-family: Tahoma, Geneva, sans-serif;
  }

  .container {
    font-family: Tahoma, Geneva, sans-serif;
  }

  .center {
    display: flex;
    justify-content: center;
    align-items: center;
    font-family: Tahoma, Geneva, sans-serif;
  }
</style>

<body class="text-center">
  <div class="center">
    <h2 class="cover-
heading">¡Bienvenido <%= Nombre %>!<img download="avatar" src="<
%=Avatar%>" width="70" height="70" style="vertical-
align:middle;margin:0px 10px">Aquí tienes tus juegos</h2>
  </div>
  <div class="center">
    </br>
    <div class="container bg-light text-dark">
      <div class="center">
        <fieldset class="field">
        <legend><svg height="13" width="15"><circle cx="8" cy="7
" r="6" fill="green" /></svg> JUEGOS ACTIVOS</legend>
        <table class="table table-striped">
        <thead>
        <tr>
        <th>Nombre</th>
        <th>Tipo</th>
        <th>Puntos</th>
        <th>Modo</th>
        </tr>
        </thead>
        <tbody>
```

```
        <%table.forEach(function(entry) {%>
          <%if (entry.JuegoActivo == 'true') {%>
            <tr>
            <td><%=entry.NombreJuego%></td>
            <td><%=entry.Tipo%></td>
            <td><%=entry.Puntos%></td>
            <td><%=entry.Modo%></td>
            <%  if (entry.Tipo == 'Juego De Cuestionario') {%>
              <td>
              <button onclick="getCuestionario(`<%=entry.cuestio
narioId%>`, `<%=entry.grupoId%>`, `<%=entry.juegoDeCuestionarioI
d%>`, `<%=entry.idCuestionarioAlumnoJuego%>`, `<%=entry.idAlumno
%>`)" class='button buttonCuestionario' id="Cuestionario" ><i cl
ass="fa fa-pencil" style="color:white"></i></button>
              </td>
              <% }%>
              <%  if (entry.Tipo == 'Juego De Colección') {%>
              <td>
              <button onclick="getColeccion(`<%=entry.coleccionI
d%>`, `<%=entry.grupoId%>`, `<%=entry.juegoDeColeccionId%>`, `<%
=entry.idColeccionAlumnoJuego%>`, `<%=entry.idAlumno%>`)" class=
'button buttonColeccion' id="Coleccion" ><i class="fa fa-
eye" style="color:white"></i></button>
              </td>
              <% }%>
              <%  if (entry.Tipo == 'Juego De Avatar') {%>
                <td>
                <button onclick="getAvatar(`<%=entry.juegoAvatar
id%>`)" class='button buttonAvatar' id="Avatar" ><i class="fa fa
-eye" style="color:white"></i></button>
                </td>
                <% }%>
            </tr>
          <% }%>
        <%});%>
        </tbody>
        </table>
        </fieldset>
      </div>
      </br></br>
```

```
      <div class="center">
        <fieldset class="field">
        <legend><svg height="13" width="15"><circle cx="8" cy="7
" r="6" fill="red" /></svg> JUEGOS INACTIVOS</legend>
        <table class="table table-striped">
        <thead>
        <tr>
        <th>Nombre</th>
        <th>Tipo</th>
        <th>Puntos</th>
        <th>Modo</th>
        <th>Ranking</th>
        </tr>
        </thead>
        <tbody>
          <%table.forEach(function(entry) {%>
            <%if (entry.JuegoActivo == 'false') {%>
              <tr>
              <td><%=entry.NombreJuego%></td>
              <td><%=entry.Tipo%></td>
              <td><%=entry.Puntos%></td>
              <td><%=entry.Modo%></td>
              <td>
           <% if (entry.Tipo == 'Juego de Competicion Liga') {%>
                  <button onclick="getRankingCompeticion(`<%=ent
ry.id%>`)" class='button buttonRankPuntos' id="RankingCompLiga"
>Ver</button>
<% } else if (entry.Tipo == 'Juego De Competición Fórmula Uno')
{%>
                  <button onclick="getRankingCompeticionFU(`<%=e
ntry.id%>`)" class='button buttonRankPuntos' id="RankingCompFU"
>Ver</button>
             <% } else if (entry.Tipo == 'Juego De Puntos') {%>
                  <button onclick="getRankingPuntos(`<%=entry.id
%>`, `<%=entry.idAlumno%>`)" class='button buttonRankPuntos' id=
"RankingPunt" >Ver</button>
        <% } else if (entry.Tipo == 'Juego De Cuestionario') {%>
                  <button onclick="getRankingCuestionario(`<%=en
try.id%>`)" class='button buttonRankPuntos' id="RankingCuest" >V
er</button>
               <% }%>
              </td>
```

```
            <% if (entry.Tipo == 'Juego De Cuestionario') {%>
              <td><%=entry.Presentacion%></td>
            <% }%>
            </tr>
          <% }%>
        <%});%>
      </tbody>
      </table>
      </fieldset>
    </div>
    </br></br>


    <div class="center">
      <fieldset class="field">
      <legend><svg height="13" width="15"><circle cx="8" cy="7
" r="6" fill="grey" /></svg> JUEGOS INCOMPLETOS</legend>
      <table class="table table-striped">
      <thead>
      <tr>
      <th>Nombre</th>
      <th>Tipo</th>
      <th>Puntos</th>
      <th>Modo</th>
      </tr>
      </thead>
      <tbody>
      <%table.forEach(function(entry) {%>
        <%if (entry.JuegoActivo == 'null') {%>
          <tr>
          <td><%=entry.NombreJuego%></td>
          <td><%=entry.Tipo%></td>
          <td><%=entry.Puntos%></td>
          <td><%=entry.Modo%></td>
          </tr>
        <% }%>
      <%});%>
      </tbody>
      </table>
      </fieldset>
    </div>
  </div>
```

```
  </div>
  </br></br></br></br>
  <div class="center">
   <button onclick="goBack()" class='buttonb buttonBack'>Go Back
</button>
  </div>
</body>
</html>
<script>
  function goBack() {
    window.history.back();
  }
  function getRankingPuntos(myid1, myuserid1) {
    window.location.href = "/rankingPuntos/" + myid1 + "/" + myu
serid1;
  }
  function getRankingCompeticion(myid2) {
    window.location.href = "/rankingPuntos/" + myid2;
  }
  function getRankingCompeticionFU(myid3) {
    window.location.href = "/rankingPuntos/" + myid3;
  }
  function getRankingCuestionario(myid4) {
    window.location.href = "/rankingPuntos/" + myid4;
  }
  function getCuestionario(myid5, grupoId, juegoDeCuestionarioId
, juegoalumnoCuestionarioId, alumnoId) {
    window.location.href = "/Cuestionario/" + myid5 + "/" + grup
oId + "/" + juegoDeCuestionarioId + "/" + juegoalumnoCuestionari
oId + "/" + alumnoId;
  }
  function getColeccion(myid6, grupoId2, juegoDeCuoleccionId, ju
egoalumnoColeccionId, alumnoId) {
    window.location.href = "/Coleccion/" + myid6 + "/" + grupoId
2 + "/" + juegoDeCuoleccionId + "/" + juegoalumnoColeccionId + "
/" + alumnoId;
  }
  function getAvatar(myid7) {
    window.location.href = "/Avatar/" + myid7;
  }
</script>
```

En esta página se crean 3 tablas con los datos de los juegos: juegos activos, inactivos e incompletos.

En caso de tratarse de un juego inactivo con puntos, se podrá ver el ranking global. Si es un cuestionario activo, se podrá acceder para contestar a las preguntas y guardar las respuestas. Si es un juego de colección, se podrá ver la colección actual y si es un juego de avatar en activo, podremos ver nuestro avatar.

Cada función extra (ver la colección de cromos, contestar el cuestionario, etc) tiene su propia función. La forma más sencilla que encontré fue la de hacer un simple *href* y en la misma URL pasar los datos necesarios para cargar la siguiente página. Esta página luego se cargará mediante la redirección que hace el Servidor junto con los datos que extrae de la anterior URL.

7. Guía para implementar el ranking de un juego

Lo primero es crear las rutas dentro del servidor:

```
app.use('/rankingPuntos', function(req, resp, next) {
    let idPuntos = req.url.substring(1);
    let rankingList = [];
    getRankingJuegoDePuntos().subscribe(ranking => {
      let filterone12 = ranking[0].juegoDePuntosId;
      let uri2 = server  + '/api/JuegosDePuntos?filter[where][or
][0][id]=' + filterone12;
      if (ranking.length > 1) {
        for (let i = 1; i < ranking.length; i++) {
          uri2 += '&filter[where][or]' + '[' + i + ']' + '[id]='
 + ranking[i].juegoDePuntosId;
        }
      }
      let filterone13 = ranking[0].alumnoId;
      let uri3 = server  + '/api/Alumnos?filter[where][or][0][id
]=' + filterone13;
      if (ranking.length > 1) {
        for (let i = 1; i < ranking.length; i++) {
          uri3 += '&filter[where][or]' + '[' + i + ']' + '[id]='
 + ranking[i].alumnoId;
        }
      }
      getListaAlumnosJuegoPuntos(uri3).subscribe(listaalumnos =>
 {
        let listaAlumnosJuegoPuntos = listaalumnos[0];
        getJuegoDePuntos(uri2, 1).subscribe(allinfo => {
          for (let j = 0; j < allinfo.length; j++) {
            rankingList.push({nombre: [allinfo[j].NombreJuego],
juegoDePuntosId: [ranking[j].juegoDePuntosId],
              alumnoId: [ranking[j].alumnoId], Puntos: [ranking[
j].PuntosTotalesAlumno]});
          }
          rankingList = rankingList.sort((a, b) => (a.Puntos > b
.Puntos) ? -1 : 1);
          resp.render('..\\..\\views\\ranking-
puntos.ejs', {table: rankingList, myId: idPuntos, listaAlumnos:
listaAlumnosJuegoPuntos});
        });
```

```
      });
    });
  });
```

En resumen, hago una consulta para obtener toda la lista de los juegos de puntos *getRankingJuegoDePuntos*, luego hago una para obtener el los datos del juego de puntos del alumno y, por último, obtengo el nombre de cada juego. Tras esto, ordenamos la lista con el ranking, pero solo a partir de los puntos y en orden decreciente. Estos datos se los pasamos luego a la página ejs para que los muestre.

Las funciones que se utilizan son las siguientes:

```
function getRankingJuegoDePuntos(juegoDePuntosId) {
  const rankingJuegoDePuntosObservable = new Observable(obs => {
    let rankingJuegoDePuntos;
    let myuri = server  + '/api/AlumnoJuegosDePuntos?filter[wher
e][or][0][juegoDePuntosId]=' + juegoDePuntosId;
    request.get(myuri, {json: true}, (error, response, body8) =>
 {
      rankingJuegoDePuntos = body8;
      obs.next(rankingJuegoDePuntos);
    });
  });
  return rankingJuegoDePuntosObservable; }
```

Simplemente obtenemos los puntos de los alumnos en función del *id* del juego de Puntos. Este *id* lo pasamos al llamar a la función.

```
function getListaAlumnosJuegoPuntos(myuri) {
  const listaAlumnosPuntosbservable = new Observable(obs => {
    let listaAlumnosJuegoDePuntos = [];
    request.get(myuri, {json: true}, (error, response, body2) =>
 {
      listaAlumnosJuegoDePuntos.push(body2);
      let size2 = body2.length;
      listaAlumnosJuegoDePuntos.push(size2);
      obs.next(listaAlumnosJuegoDePuntos);
    });
  });
  return listaAlumnosPuntosbservable;
}
```

Esta función se utiliza para saber los nombres de los alumnos que juegan a dicho juego de Puntos.

La función *getJuegoDePuntos()* está explicada en la anterior guía del **Anexo**, en el apartado 6.5.

En cuanto a la página ejs, está definida de la siguiente forma:

```
<body class="text-center">
<div class="cover-container d-flex w-100 h-100 p-3 mx-auto flex-
column">
<html>
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/l
ibs/font-awesome/4.7.0/css/font-awesome.min.css">
<style>
  td, th {
  border: 1px solid #dddddd;
  text-align: left;
  padding: 8px;
  font-family: Tahoma, Geneva, sans-serif;
  }

  tr:nth-child(even) {
  background-color: #d8d9d9;
  font-family: Tahoma, Geneva, sans-serif;
  }

  .buttonb {
    padding: 5px;
    background-color: #0f2d4b;
    border: 0;
    box-sizing: border-box;
    cursor: pointer;
    font-weight: bold;
    color: #ffffff;
    font-family: Tahoma, Geneva, sans-serif;
  }

  .buttonBack:hover {
    background-color: #952807;
    color: white;
    font-family: Tahoma, Geneva, sans-serif;
  }

  .login-form input[type="submit"] {
```

```
    width: 100%;
    padding: 5px;
    background-color: #002d5b;
    border: 0;
    box-sizing: border-box;
    cursor: pointer;
    font-weight: bold;
    color: #ffffff;
    font-family: Tahoma, Geneva, sans-serif;
  }

  .container {
    font-family: Tahoma, Geneva, sans-serif;
  }

  .center {
    display: flex;
    justify-content: center;
    align-items: center;
    font-family: Tahoma, Geneva, sans-serif;
  }

</style>
<body class="text-center" onload="myAlert()">
  <div class="center">
    </br>
    <div class="container bg-light text-dark">
      <div class="center">
        <fieldset class="field">
        <legend> <svg height="13" width="15"><circle cx="8" cy="
7" r="6" fill="blue" /></svg> Ranking del juego<b><i>"<%=table[0
].nombre%>"</i></b></legend>
        <table class="table table-striped">
        <thead>
        <tr>
        <th>Jugador</th>
        <th>Puntos</th>
        </tr>
        </thead>
        <tbody>
        <%table.forEach(function(entry) {%>
            <tr>
```

```
              <%listaAlumnos.forEach(function(entry2) {%>
                <%if (entry2.id == entry.alumnoId) {%>
                <td>
                  <%if (myId == entry.alumnoId) {%>
                  <i class="fa fa-
star checked" style="color:yellow"></i>
                   <% }%>
                <%=entry2.Nombre + " " + entry2.PrimerApellido%>
                </td>
                <% }%>
                <%});%>
            <td><%=entry.Puntos%></td>
            </tr>
        <%});%>
        </tbody>
        </table>
        </fieldset>
      </div>
    </div>
  </div>
  </br></br></br></br>
  <div class="center">
    <button onclick="goBack()" class='buttonb buttonBack'>Go Bac
k</button>
  </div>
</body>
</html>
<script>
  function myAlert() {
    alert("Aquí puedes ver el ranking del juego de Puntos selecc
ionado.\n\nTu puesto del ranking está marcado con una estrella")
;
  }
  function goBack() {
    window.history.back();
  }
</script>
```

Con los datos que se obtienen del servidor se crea el ranking ordenado por puntos.

Es una tabla que vamos iterando para ir mostrando los datos que queremos visualizar: el nombre del alumno y los puntos que tiene. Como la lista que

obtenemos ya está ordenada por puntos, no necesitamos hacer nada más. Como detalle he añadido una estrella al lado del nombre del alumno que está viendo el ranking:

```
<%if (myId == entry.alumnoId) {%>
<i class="fa fa-star checked" style="color:yellow"></i>
```

La forma en la que lo he hecho es comparando el id de la tabla con el id que obtenemos directamente del servidor.

También, nada más cargar la página, lo primero que se muestra es una alerta al usuario:

```
<body class="text-center" onload="myAlert()">
```

La función la tenemos en el apartado de scripts:

```
function myAlert() {
 alert("Aquí puedes ver el ranking del juego de Puntos seleccion
ado.\n\nTu puesto del ranking está marcado con una estrella");}
```

8. Guía para implementar el ver/contestar un juego de cuestionario

Primero de todo mostraré la configuración para la ruta:

```
  app.use('/Cuestionario', function(req, resp, next) {
    let idCuest = req.url.split('/');
    getCuestionario(idCuest[1], idCuest[2], idCuest[3], idCuest[
4], idCuest[5]).subscribe(response => {
      Cuestionarios = response;
      resp.render('..\\..\\views\\game-
cuestionario.ejs', {table: Cuestionarios, Nombre: nombreUser,
        numeroPreguntas: Cuestionarios.length});
    });
  });
```

Se puede ver que es bastante simple. En este caso obtengo la información de necesaria de la *url* de la petición realizada en la página del listado de juegos.

La función para extraer los datos de la base de datos es la siguiente:

```
function getCuestionario(idCuestionario, idGrupo, idJuegoDeCuest
ionario, idJuegoDeCuestionarioAlumno, idAlumno) {
  const CuestionarioObservable = new Observable(obs => {
    let pregCuestInfo = [];
    let preguntasyCuestionario = [];
    let pregunta = [];
    let cuestionario = [];
    let preguntasCuestionarioList = [];
    request.get(server + '/api/PreguntasDelCuestionario?filter[w
here][cuestionarioId]=' + idCuestionario,
    {json: true}, (error, response, body) => {
      preguntasyCuestionario = body;
      let uri = '';
      let size = Object.keys(body).length;
      let filterone2 = preguntasyCuestionario[0].cuestionarioId;
      uri = server + '/api/Cuestionarios?filter[where][or][0][id
]=' + filterone2;
      if (size > 1) {
        for (let k = 1; k < size; k ++) {
          if (preguntasyCuestionario[k -
 1].cuestionarioId != preguntasyCuestionario[k].cuestionarioId)
{
```

```
            uri += '&filter[where][or]' + '[' + k + ']' + '[id]=
' + preguntasyCuestionario[k].cuestionarioId;
          }
        }
      }
      let uri2 = '';
      let filterone = preguntasyCuestionario[0].preguntaId;
      uri2 = server + '/api/Preguntas?filter[where][or][0][id]='
 + filterone;
      if (size > 1) {
        for (let i = 1; i < size; i++) {
          uri2 += '&filter[where][or]' + '[' + i + ']' + '[id]='
 + preguntasyCuestionario[i].preguntaId;
        }
      }
    request.get(uri, {json: true}, (error, response, body2) => {
        cuestionario = body2;
        let sizeCuestionario = body2.length;
   request.get(uri2, {json: true}, (error, response, body3) => {
          pregunta = body3;
          let sizePreguntas = body3.length;
          let preguntaIndiv = [];
          if (sizePreguntas > 0) {
            for (let l = 0; l < sizeCuestionario; l++) {
              if (sizePreguntas > 1) {
                for (let j = 0; j < sizePreguntas; j++) {
                  preguntaIndiv.push({TituloP: pregunta[j].Titul
o, Pregunta: pregunta[j].Pregunta, Tema: pregunta[j].Tematica,
                    RespuestaCorrecta: pregunta[j].RespuestaCorr
ecta, RespuestaIncorrecta1: pregunta[j].RespuestaIncorrecta1,
                    RespuestaIncorrecta2: pregunta[j].RespuestaI
ncorrecta2, RespuestaIncorrecta3: pregunta[j].RespuestaIncorrect
a3,
                    FeedbackCorrecto: pregunta[j].FeedbackCorrec
to, FeedbackIncorrecto: pregunta[j].FeedbackIncorrecto,
                    profesorId: pregunta[j].profesorId, idPregun
ta: pregunta[j].id});
                }
              }
              pregCuestInfo.push({TituloC: [cuestionario[l].Titu
lo], DescripcionC: [cuestionario[l].Descripcion],
```

```
                idCuestionarioC: [cuestionario[l].id], profesorI
dC: [cuestionario[l].profesorId], Preguntas: [preguntaIndiv], Gr
upoId: [idGrupo],
                JuegoDeCuestionarioId: [idJuegoDeCuestionario],
JuegoDeCuestionarioAlumnoId: [idJuegoDeCuestionarioAlumno],
                idAlumno: [idAlumno]});
            }
          }
          obs.next(pregCuestInfo);
        });
      });
    });
  });
  return CuestionarioObservable;
}
```

En esta ocasión se hacen 3 peticiones al servidor:

```
request.get(server + '/api/PreguntasDelCuestionario?filter[where
][cuestionarioId]=' + idCuestionario,
    {json: true}, (error, response, body) => {
```

Esta petición nos da las preguntas del cuestionario a partir del id del cuestionario.

```
request.get(uri, {json: true}, (error, response, body2) => {
request.get(uri2, {json: true}, (error, response, body3) => {
```

Los parámetros *uri* y *uri2* los obtenemos tras procesar un poco la información que se obtiene de la primera consulta:

```
      let uri = '';
      let size = Object.keys(body).length;
      let filterone2 = preguntasyCuestionario[0].cuestionarioId;
      uri = server + '/api/Cuestionarios?filter[where][or][0][id
]=' + filterone2;
      if (size > 1) {
        for (let k = 1; k < size; k ++) {
          if (preguntasyCuestionario[k -
 1].cuestionarioId != preguntasyCuestionario[k].cuestionarioId)
{
            uri += '&filter[where][or]' + '[' + k + ']' + '[id]=
' + preguntasyCuestionario[k].cuestionarioId;
          }
```

```
        }
      }
```

Creamos la siguiente *url* (*uri*) para hacer la consulta, a partir de los id de cuestionarios que tenga asignado el alumno. Esta consulta nos sirve para obtener ciertos datos del cuestionario: nombre, preguntas que contiene, etc.

```
      let uri2 = '';
      let filterone = preguntasyCuestionario[0].preguntaId;
      uri2 = server + '/api/Preguntas?filter[where][or][0][id]='
+ filterone;
      if (size > 1) {
        for (let i = 1; i < size; i++) {
          uri2 += '&filter[where][or]' + '[' + i + ']' + '[id]='
+ preguntasyCuestionario[i].preguntaId;
        }
      }
```

La segunda *url* también la creamos a partir de la primera consulta. En este caso filtramos las preguntas a partir del id de la pregunta que contenga el juego de cuestionario del alumno. Esta consulta se guarda en una lista individual:

```
   request.get(uri2, {json: true}, (error, response, body3) => {
          pregunta = body3;
```

A partir de aquí, recorremos los datos que haya y los guardamos en otra lista con un cierto nombre de clave para cada valor:

```
preguntaIndiv.push({TituloP: pregunta[j].Titulo, Pregunta: pregu
nta[j].Pregunta, Tema: pregunta[j].Tematica, RespuestaCorrecta:
pregunta[j].RespuestaCorrecta, RespuestaIncorrecta1: pregunta[j]
.RespuestaIncorrecta1,RespuestaIncorrecta2: pregunta[j].Respuest
aIncorrecta2, RespuestaIncorrecta3: pregunta[j].RespuestaIncorre
cta3, FeedbackCorrecto: pregunta[j].FeedbackCorrecto,
FeedbackIncorrecto: pregunta[j].FeedbackIncorrecto,
profesorId: pregunta[j].profesorId, idPregunta: pregunta[j].id})
;
```

Esta lista luego se introduce como valor para otra lista que contiene toda la información necesaria (tanto del cuestionario como de las preguntas):

```
pregCuestInfo.push({TituloC: [cuestionario[l].Titulo], Descripci
onC: [cuestionario[l].Descripcion], idCuestionarioC: [cuestionar
io[l].id], profesorIdC: [cuestionario[l].profesorId], Preguntas:
 [preguntaIndiv], GrupoId: [idGrupo], JuegoDeCuestionarioId: [id
```

```
JuegoDeCuestionario], JuegoDeCuestionarioAlumnoId: [idJuegoDeCue
stionarioAlumno], idAlumno: [idAlumno]});
```

Tras esto, ya tenemos todos los datos necesarios para procesarlos y mostrarlos en pantalla.

A la página ejs le pasaremos estos datos y realizaremos lo siguiente:

```
<body class="text-center">
<div class="cover-container d-flex w-100 h-100 p-3 mx-auto flex-
column">
<html>

<style>
  .b1
  {
    position:absolute;
  }

  .b2
  {
    position:absolute;
    left : 100px;
  }

  td, th {
    border: 1px solid #dddddd;
    text-align: left;
    padding: 8px;
    font-family: Tahoma, Geneva, sans-serif;
  }

  tr:nth-child(even) {
    background-color: #dddddd;
    font-family: Tahoma, Geneva, sans-serif;
  }

  .container {
    font-family: Tahoma, Geneva, sans-serif;
  }

  fieldset {
    background-color: #ffffff;
```

```
  }

  legend {
    background-color: #224475;
    color: white;
    padding: 5px 10px;
    font-family: Tahoma, Geneva, sans-serif;
  }

  .cover-container {
    display: inline-block;
  }

  .disable {
    pointer-events: none;
    opacity: 0.4;
}

</style>

<body class="text-center">
  <div class="cover-container">
    <fieldset class="field">
    <legend>Cuestionario "<%=table[0].TituloC%>"</legend>
    <%let y = 1;%>
    <%let correctAnswers = [];%>
    <%table.forEach(function(entry) {%>
      <%for (let z = 0; z < entry.Preguntas.length; z++) {%>
        <div id="quiz">
          <form id="quizQuestion" name="quizQuestion<%=z%>">
            <h4>Pregunta <%=y%>: <%=entry.Preguntas[z].Pregunta%
></h4>
            <div id="q"><input type="radio" name="q" value="<%=e
ntry.Preguntas[z].RespuestaCorrecta%>"><%=entry.Preguntas[z].Res
puestaCorrecta%></div>
            <div id="q"><input type="radio" name="q" value="<%=e
ntry.Preguntas[z].RespuestaIncorrecta1%>"><%=entry.Preguntas[z].
RespuestaIncorrecta1%></div>
            <div id="q"><input type="radio" name="q" value="<%=e
ntry.Preguntas[z].RespuestaIncorrecta2%>"><%=entry.Preguntas[z].
RespuestaIncorrecta2%></div>
```

```
            <div id="q"><input type="radio" name="q" value="<%=e
ntry.Preguntas[z].RespuestaIncorrecta3%>"><%=entry.Preguntas[z].
RespuestaIncorrecta3%></div>
            <%correctAnswers.push(entry.Preguntas[z].RespuestaCo
rrecta);%>
          </form>
        </div>
        <%y++;%>
      <%} %>
    <%});%>
      </fieldset>
  </div>
  </br></br>
  <button onclick="cancel()" class='b2'>Cancel</button>
  <button onclick="checkResults()" class='b1'>Answer</button>
</body>
</html>
  <script>
  var numQuestions = `<%=table[0].Preguntas.length%>`;
  var formsDoc = document.forms;
  var minimumChecked = false;
  var answersFinal = "";
  var correctAnswersList = `<%=correctAnswers%>`.split(",");

  function loopInfo(numberOfQuestions, formsDocuments) {
    var buttonsPushed = 0;
    var answers = [];
    var checked = [];

    for (var o = 0; o < numberOfQuestions; o++) {
      for (var p = 0; p < formsDocuments[o].elements.length; p++
) {
        var inputElements = formsDocuments[o].getElementsByTagNa
me('input');
        if (inputElements[p].checked) {
          buttonsPushed++;
        }
        console.log(formsDocuments[o].elements["q"].value);
    }
    checked.push(buttonsPushed);
    answers.push(formsDocuments[o].elements["q"].value.trim());
  }
```

```
    answersFinal = answers;
    return buttonsPushed;
  }

  function checkResults() {
    var buttonIsPushed = loopInfo(numQuestions, formsDoc);
    if (buttonIsPushed == numQuestions){
      for (var o = 0; o < numQuestions; o++) {
      for (var p = 0; p < formsDoc[o].elements.length; p++) {
        var inputElements = formsDoc[o].getElementsByTagName('in
put');
        inputElements[p].disabled = true;
      }
     }
    }
    if (buttonIsPushed == answersFinal.length){
      var myAnswerScores = scoreCalc();
      window.location.href = "/answerQuiz/" + `<%=table[0].idCue
stionarioC%>` + "/" + `<%=table[0].GrupoId%>` + myAnswerScores +
 "/" + `<%=table[0].JuegoDeCuestionarioAlumnoId%>` + "/" + `<%=t
able[0].idAlumno%>`;
    }
    else {
      alert("¡Necesitas contestar todas las preguntas!");
    }
  }
  function scoreCalc(){
    var myAnswers = "";
    var scoresOK = 0;
    var scoresKO = 0;
    var scores = "";
    for (var q = 0; q < answersFinal.length; q++){
      myAnswers += "/" + answersFinal[q];
      if (answersFinal[q] == correctAnswersList[q]) {
        scoresOK++;
        } else {
          scoresKO++;
        }
      }
    scores = "/" + scoresOK + "/" + scoresKO + "/" + `<%=table[0
].JuegoDeCuestionarioId%>`;
    return scores;
```

```
  }
  function cancel() {
    window.history.back();
  }
  </script>
```

A cada respuesta de la pregunta se le asigna el id “q”:

```
<div id="q"><input type="radio" name="q" value="<%=entry.Pregunt
as[z].RespuestaCorrecta%>"><%=entry.Preguntas[z].RespuestaCorrec
ta%></div>
```

Esto nos servirá para saber si ha sido seleccionada o no.

Para saber si se ha seleccionado por lo menos una respuesta para cada pregunta, se tiene que implementar la siguiente función primero:

```
function loopInfo(numberOfQuestions, formsDocuments) {
    var buttonsPushed = 0;
    var answers = [];
    var checked = [];

    for (var o = 0; o < numberOfQuestions; o++) {
      console.log(formsDocuments[o]);
      for (var p = 0; p < formsDocuments[o].elements.length; p++
) {
        var inputElements = formsDocuments[o].getElementsByTagNa
me('input');
        if (inputElements[p].checked) {
          buttonsPushed++;
        }
        console.log(formsDocuments[o].elements["q"].value);
    }
    checked.push(buttonsPushed);
    answers.push(formsDocuments[o].elements["q"].value.trim());
  }
    answersFinal = answers;
    return buttonsPushed;
  }
```

Consultamos el número total de preguntas y luego miramos en cada uno de los elementos de tipo “input” (que son las preguntas), si ha sido seleccionado o no.

En caso que haya alguna respuesta seleccionada, se añade 1 unidad al recuento de preguntas contestadas (*buttonsPushed*).

Con todo esto devolvemos el recuento de preguntas contestadas y añadimos la respuesta que haya sido seleccionada una lista (*answersFinal*).

Esta función la llamamos desde la función principal:

```
function checkResults() {
    var buttonIsPushed = loopInfo(numQuestions, formsDoc);
    if (buttonIsPushed == numQuestions){
      for (var o = 0; o < numQuestions; o++) {
      for (var p = 0; p < formsDoc[o].elements.length; p++) {
        var inputElements = formsDoc[o].getElementsByTagName('in
put');
        inputElements[p].disabled = true;
      }
     }
    }
    if (buttonIsPushed == answersFinal.length){
      var myAnswerScores = scoreCalc();
      window.location.href = "/answerQuiz/" + `<%=table[0].idCue
stionarioC%>` + "/" + `<%=table[0].GrupoId%>` + myAnswerScores +
 "/" + `<%=table[0].JuegoDeCuestionarioAlumnoId%>` + "/" + `<%=t
able[0].idAlumno%>`;
    }
    else {
      alert("¡Necesitas contestar todas las preguntas!");
    }
  }
```

Tras llamar al recuento de preguntas contestadas, deshabilitamos la opción de poder cambiar de pregunta y finalmente calculamos el número de preguntas correctas e incorrectas en otra función llamada “*scoreCal()*”. Lo único que queda es enviar al servidor las respuestas junto a los datos necesarios para poder guardarlas. En caso de no haberlas contestado todas, alertaremos al alumno de que tiene que hacerlo.

La última función es la que nos permite calcular las respuestas correctas e incorrectas:

```
function scoreCalc(){
    var myAnswers = "";
    var scoresOK = 0;
    var scoresKO = 0;
```

```
    var scores = "";
    for (var q = 0; q < answersFinal.length; q++){
      myAnswers += "/" + answersFinal[q];
      if (answersFinal[q] == correctAnswersList[q]) {
        scoresOK++;
        } else {
          scoresKO++;
        }
      }
    scores = "/" + scoresOK + "/" + scoresKO + "/" + `<%=table[0
].JuegoDeCuestionarioId%>`;
    return scores;
  }
```

A partir de las respuestas que guardamos con la función de *loopInfo()*, podemos comparar si la respuesta es correcta o no haciendo uso de la lista de respuestas correctas. Esta la obtenemos al principio:

```
<%let correctAnswers = [];%>
```

Y se rellena mientras recorremos la lista de preguntas y vamos mostrando en pantalla las respuestas:

```
<%correctAnswers.push(entry.Preguntas[z].RespuestaCorrecta);%>
```

Dependiendo de si la respuesta es correcta o no, sumamos 1 unidad a la lista de respuestas correctas (socresOK) o incorrectas (scoresKO).
Finalmente, devolvemos una cadena de texto con el total de preguntas correctas e incorrectas. Esta cadena la ponemos de tal manera que encaje dentro de la *url* que se envía al servidor.

Tal y como hemos visto con anterioridad:

```
window.location.href = "/answerQuiz/" + `<%=table[0].idCuestiona
rioC%>` + "/" + `<%=table[0].GrupoId%>` + myAnswerScores + "/" +
 `<%=table[0].JuegoDeCuestionarioAlumnoId%>` + "/" + `<%=table[0
].idAlumno%>`;
```

Las respuestas se envían al servidor utilizando la url que empieza con */answerQuiz*.

Dentro del servidor, parseamos la *url* completa y sacamos cada uno de los datos necesarios:

```
  app.use('/answerQuiz', function(req, resp, next) {
    let myValues = req.url.split('/answerQuiz').pop();
```

```
    let singleValues = myValues.split('/');
    let cuestionarioId = singleValues[1];
    let grupoId = singleValues[2];
    let puntosCorrectos = parseInt(singleValues[3]);
    let puntosIncorrectos = parseInt(singleValues[4]);
    let idCuestionarioJuego = singleValues[5];
    let puntosTotales = (puntosCorrectos + puntosIncorrectos);
```

El primer valor después de cortar cada variable estará siempre vacío, por lo que empezaremos desde la posición 1.

Tras esto, calculamos el total de puntos, añadimos una comprobación en caso de que la nota sea inferior a 0, y calculamos la nota final teniendo en cuenta que solo queremos 2 cifras decimales:

```
let puntosCorrectosTotales = (puntosCorrectos -
(puntosIncorrectos / 3));
    let idJuegoCuestionarioAlumno = singleValues[6];
    let idAlumno = singleValues[7];
    miNota = puntosCorrectosTotales;
    if (miNota <= 0) {
      miNota = 0;
    }
    miNota = miNota / puntosTotales;
    miNota = (miNota * 10).toFixed(2);
```

Una vez tenemos la nota calculada, la podemos enviar a la base de datos. En este caso no solo tenemos que modificar/añadir la nota, sino que también tenemos que cambiar el número de respuestas correctas e incorrectas del alumno:

```
request.get(server + '/api/JuegosDeCuestionario?filter[where][o
r][0][id]=' + idCuestionarioJuego, {json: true}, (error, respons
e, body) => {
      JuegosDeCuestionario = JSON.stringify({NombreJuego: body.N
ombreJuego, PuntuacionCorrecta: puntosCorrectos,
        PuntuacionIncorrecta: puntosIncorrectos, Presentacion: b
ody.Presentacion, JuegoActivo: false,
        JuegoTerminado: body.JuegoTerminado, profesorId: body.pr
ofesorId, grupoId: grupoId, cuestionarioId: cuestionarioId});
        alumnoJuegoDeCuestionario = JSON.stringify({Nota: (miNot
a).toString(), alumnoId: idAlumno,
          juegoDeCuestionarioId: idCuestionarioJuego});
```

```
        guardarRespuestas(idCuestionarioJuego, grupoId, idJuegoC
uestionarioAlumno);
        guardarNota(idJuegoCuestionarioAlumno);
        getListaJuegos(userId).subscribe(response => {
          Juegos = response;
          resp.render('..\\..\\views\\games.ejs', {Nombre: nombr
eUser, table: Juegos, Avatar: miAvatar});
        });
      });
    });
```

Guardamos los datos del cuestionario dentro de la variable global *JuegosDeCuestionario* y los datos del juego de cuestionario del alumno en la varible, también global, *alumnoJuegoDeCuestionario*.

Antes de explicar las 2 funciones que nos permiten guardar los datos actualizados, tenemos que fijarnos que al final volvemos a cargar la página con la lista de juegos del alumno.

Las funciones que nos permiten guardar los datos son las siguientes:

```
function guardarRespuestas(idJuegoCuestionario, idGrupo, idAlumn
oJuegoDeCuestionario) {
  let url = server + '/api/Grupos/' + idGrupo + '/juegosDeCuesti
onario/' + idJuegoCuestionario;
  request({url: url, method: 'PUT', json: JSON.parse(JuegosDeCue
stionario)}, function(error, request, body) {
  }); }
```

```
function guardarNota(idAlumnoJuegoDeCuestionario) {
  let url2 = server  + '/api/AlumnosJuegoDeCuestionario/' + idAl
umnoJuegoDeCuestionario;
  request({url: url2, method: 'PUT', json: JSON.parse(alumnoJueg
oDeCuestionario)}, function(error2, request2, body2) {
  }); }
```

Al tener que actualizar los datos, hacemos uso de la petición PUT. Es por ello que hacemos uso de las dos variables que antes habíamos guardado: *JuegosDeCuestionario* y *alumnoJuegoDeCuestionario*. En caso de tener una colección con los datos que pasamos, la creará, por lo que también nos sirve para guardar datos nuevos.

9. Guía para implementar la visualización de la colección de cromos

Primero de todo mostraré la configuración para la ruta:

```
app.use('/Coleccion', function(req, resp, next) {
    let idCol = req.url.split('/');
    let coleccionAlumno, coleccion;
    getColeccion(idCol[1], idCol[2], idCol[3], idCol[4], idCol[5
]).subscribe(response => {
      coleccion = response[0];
      coleccionAlumno = response[1];
      for (let z = 0; z < coleccionAlumno.length; z++) {
        for (let y = 0; y < coleccionAlumno.length; y++) {
          if (JSON.stringify(coleccion[z].idCColeccion) == JSON.
stringify(coleccionAlumno[y].cromoIdCColeccionAlumno)) {
            y++;
          }
        }
      }
      resp.render('..\\..\\views\\game-
coleccion.ejs', {Coleccion: coleccion, ColeccionAlumno: coleccio
nAlumno, Nombre: nombreUser});
    });
  });
```

Podemos ver que realizamos la consulta en la base de datos y obtenemos dos colecciones, una completa (Colección) y otra solo con los cromos del alumno (ColeccionAlumno).

La función que gestiona la consulta a la base de datos es la siguiente:

```
function getColeccion(idColeccion, idGrupo, idJuegoDeColeccion,
idJuegoDeColeccionAlumno, idAlumno) {
  const ColeccionObservable = new Observable(obs => {
    let albumAlumnoInfo = [];
    let datosAlbum = [];
    let datosCromos = [];
    request.get(server + '/api/Cromos?filter[where][coleccionId]
=' + idColeccion,
    {json: true}, (error, response, body) => {
      datosCromos = body;
      let sizeCromos = Object.keys(body).length;
```

```
      request.get(server  + '/api/Albumes?filter[where][alumnoJu
egoDeColeccionId]=' + idJuegoDeColeccionAlumno, {json: true}, (e
rror, response, body2) => {
        datosAlbum = body2;
        let sizeColeccion = body2.length;
        let cromosAlumno = [];
        let cromosTotal = [];
        if (sizeCromos > 0) {
          for (let j = 0; j < sizeCromos; j++) {
            cromosTotal.push({NombreCColeccion: [datosCromos[j].
Nombre], ProbabilidadCColeccion: [datosCromos[j].Probabilidad],
NivelCColeccion: [datosCromos[j].Nivel],
              idCColeccion: [datosCromos[j].id], coleccionIdCCol
eccion: [datosCromos[j].coleccionId],
              ImagenCColeccionDelante: [datosCromos[j].ImagenDel
ante], ImagenCColeccionDetras: [datosCromos[j].ImagenDetras]});
          }
          for (let l = 0; l < sizeColeccion; l++) {
            cromosAlumno.push({idCColeccionAlumno: [datosAlbum[l
].id], cromoIdCColeccionAlumno: [datosAlbum[l].cromoId],
              alumnoJuegoDeColeccionIdCColeccionAlumno: [datosAl
bum[l].alumnoJuegoDeColeccionId]});
          }
          albumAlumnoInfo.push(cromosTotal, cromosAlumno);
        }
        obs.next(albumAlumnoInfo);
      });
    });
  });
  return ColeccionObservable;
```

En esta función hacemos dos consultas diferentes:

- La primera para saber todos los cromos de una colección en concreto:

```
request.get(server + '/api/Cromos?filter[where][coleccionId]
=' + idColeccion, {json: true}, (error, response, body) => {
```

El id de la colección se lo habíamos pasado con anterioridad a la función.

- La segunda consulta es para saber los cromos que tiene el alumno:

```
request.get(server  + '/api/Albumes?filter[where][alumnoJu
```

```
egoDeColeccionId]=' + idJuegoDeColeccionAlumno, {json: true}
, (error, response, body2) => {
```

De la misma manera que antes, el id del juego de colección, se lo pasamos al llamar a la función.

A diferencia de otras funciones, en este caso creamos una lista para cada consulta:

```
let datosAlbum = [];
let datosCromos = [];
```

Entonces las guardamos dentro de otra lista para, finalmente, devolverla al usuario:

```
albumAlumnoInfo.push(cromosTotal, cromosAlumno);
```

A la hora de rellenar cada lista, simplemente recorremos la lista (de cromos o del álbum) y guardamos cada uno de los datos con una clave determinada. Esto siempre que la lista de cromos contenga algo:

```
if (sizeCromos > 0) {
     for (let j = 0; j < sizeCromos; j++) {
           cromosTotal.push({NombreCColeccion: [datosCromos[j].
Nombre], ProbabilidadCColeccion: [datosCromos[j].Probabilidad],
NivelCColeccion: [datosCromos[j].Nivel],
             idCColeccion: [datosCromos[j].id], coleccionIdCCol
eccion: [datosCromos[j].coleccionId],
             ImagenCColeccionDelante: [datosCromos[j].ImagenDel
ante], ImagenCColeccionDetras: [datosCromos[j].ImagenDetras]});
          }
```

Esto nos permitirá, más adelante, acceder a cada valor mediante el nombre de clave.

Tras esto, ya lo podemos mostrar en pantalla:

```
<html>
<body class="text-center">
<div class="cover-container d-flex w-100 h-100 p-3 mx-auto flex-
column">
<html>

<style>
  .b2
  {
```

```
    position:absolute;
    left : 100px;
  }

  td, th {
  border: 1px solid #dddddd;
  text-align: left;
  padding: 8px;
  font-family: Tahoma, Geneva, sans-serif;
  }

  tr:nth-child(even) {
  background-color: #62d1d1;
  font-family: Tahoma, Geneva, sans-serif;
  }

  .container {
    font-family: Tahoma, Geneva, sans-serif;
  }

  fieldset {
    background-color: #ffffff;
  }

  legend {
    background-color: #224475;
    color: white;
    padding: 5px 10px;
    font-family: Tahoma, Geneva, sans-serif;
  }

  .cover-container {
    display: inline-block;
  }

  .disable {
    pointer-events: none;
    opacity: 0.4;
}

</style>
<body class="text-center">
```

```
  <div class="center">
    <fieldset class="field">
    <legend><svg height="13" width="15"><circle cx="8" cy="7" r=
"6" fill="grey" /></svg> Album </legend>
    <table class="table table-striped">
    <tbody>
      <tr>
    <%let myCromos = [];%>
    <%for (let z = 0; z < Coleccion.length; z++) {%>
        <td><img id="<%=z%>" download="cromo" src="http://localh
ost:3000/api/imagenes/ImagenCromo/download/<%=Coleccion[z].Image
nCColeccionDelante%>" onmouseover="this.src='http://localhost:30
00/api/imagenes/ImagenCromo/download/<%=Coleccion[z].ImagenCCole
ccionDetras%>';" onmouseout="this.src='http://localhost:3000/api
/imagenes/ImagenCromo/download/<%=Coleccion[z].ImagenCColeccionD
elante%>';" width="100" style="vertical-
align:middle;margin:0px 10px;filter: grayscale(100%);"></td>
        <%for (let y = 0; y < ColeccionAlumno.length; y++) {%>
          <% if (JSON.stringify(Coleccion[z].idCColeccion) == JS
ON.stringify(ColeccionAlumno[y].cromoIdCColeccionAlumno)) {%>
            <%myCromos.push(z);%>
            <body onload="gotIt(`<%=z%>`)"></body>
            <%y++;%>
          <% }%>
        <%}%>
    <%}%>
  </tr>
    </tbody>
    </table>
    </fieldset>
  </div>
  </div>
  </br></br>
  <button onclick="cancel()" class='b2'>Atrás</button>
</body>
</html>
<script>
  function gotIt(myId) {
    document.getElementById(myId).style.filter = "grayscale(0%)"
;
  }
  function cancel() {
```

```
    window.history.back();
  }
</script>
```

Como se puede observar, mostramos cada cromo y le añadimos un id propio, a partir del valor de z:

```
<td><img id="<%=z%>" download="cromo" src="http://localhost:3000
/api/imagenes/ImagenCromo/download/<%=Coleccion[z].ImagenCColecc
ionDelante%>" onmouseover="this.src='http://localhost:3000/api/i
magenes/ImagenCromo/download/<%=Coleccion[z].ImagenCColeccionDet
ras%>';" onmouseout="this.src='http://localhost:3000/api/imagene
s/ImagenCromo/download/<%=Coleccion[z].ImagenCColeccionDelante%>
';" width="100" style="vertical-align:middle;margin:0px 10px;
filter: grayscale(100%);"></td>
```

También se añaden dos funciones: *onmouseover* y *onmouseout*. Esto permite cambiar de imagen cada vez que pasamos por encima. De esta manera se puede mostrar el reverso de la carta. Se le añade el filtro *filter: grayscale(100%)* para poder mostrarlo en gris si el alumno no tiene el cromo.

Tras esto, comprobamos si el alumno tiene o no el cromo. Si lo tiene, llamamos a la función *gotIt(idCromo)* y le pasamos el valor de z para saber en qué cromo estamos:

```
<%for (let y = 0; y < ColeccionAlumno.length; y++) {%>
          <% if (JSON.stringify(Coleccion[z].idCColeccion) == JS
ON.stringify(ColeccionAlumno[y].cromoIdCColeccionAlumno)) {%>
            <%myCromos.push(z);%>
            <body onload="gotIt(`<%=z%>`)"></body>
            <%y++;%>
          <% }%>
        <%}%>
    <%}%>
```

La función *gotIt(idCromo)* nos permite quitarle la escala de grises que se le asigna a cada cromo nada más se muestra en pantalla:

```
function gotIt(myId) {
document.getElementById(myId).style.filter = "grayscale(0%)";
  }
```

10. Guía para implementar la visualización del Avatar

Lo primero es crear la nueva ruta:

```
app.use('/Avatar', function(req, resp, next) {
    console.log('MY HOST avatar IS: ' + req.url);
    let MYURL = req.url.split('/');
    let idJuegoAvatarAlumno = MYURL[1];
    getAvatar(idJuegoAvatarAlumno).subscribe(response => {
      let miJuegoAvatar = response;
      resp.render('..\\..\\views\\game-
avatar.ejs', {Nombre: nombreUser, table: miJuegoAvatar});
    });
  });
```

Se puede observar que se utiliza la función *getAvatar(idJuegoAvatarAlumno)*. Esta función coge los datos necesarios de la colección del juego de Avatar del alumno a partir de un id. Este id es el del juego de Avatar del alumno.
La función es la siguiente:

```
function getAvatar(idAlumnojJuegoAvatar) {
 const juegoAvatarObservable = new Observable(obs => {
 let alumnoJuegoAvatar = [];
 let myuri = server + '/api/alumnosJuegoAvatar?filter[where][jue
goDeAvatarId]=' + idAlumnojJuegoAvatar;
 request.get(myuri, {json: true}, (error, response, body12) => {
      alumnoJuegoAvatar = body12;
      obs.next(alumnoJuegoAvatar);
    });
  });
  return juegoAvatarObservable;
}
```

En esta función simplemente guardamos toda la respuesta a la consulta de *alumnosJuegoAvatar* y devolvemos un observable. Con esto nos podemos asegurar de que, al suscribirnos, esperaremos a tener una respuesta para seguir con las siguientes tareas.

Tras esto, podemos implementar la página que mostrará el avatar del alumno:

```
<body class="text-center">
<div class="cover-container d-flex w-100 h-100 p-3 mx-auto flex-
column">
<html>
```

```
<style>
  td, th {
  border: 1px solid #dddddd;
  font-family: Tahoma, Geneva, sans-serif;
  color: white;
  text-align: left;
  padding: 8px;
  }

  tr:nth-child(even) {
  background-color: #dddddd;
  font-family: Tahoma, Geneva, sans-serif;
  }

  .cover-container {
    display: inline-block;
    margin-right: 3%;
  }

  .avatarBase {
    position: relative;
  }

  .avatar {
    position: absolute;
    top: 45;
    left: 10;
  }
</style>

<body class="text-center" style="display:flex">
  <div class="cover-container">
    <legend style="font-family: Tahoma, Geneva, sans-
serif;"><svg height="13" width="15"><circle cx="8" cy="7" r="6"
fill="black" /></svg> Mi Avatar</legend>
    </br>
    <%table.forEach(function(entry) {%>
      <img class="avatarBase" src="http://localhost:3000/api/ima
genes/ImagenesAvatares/download/<%=entry.Silueta%>">
      <% if (entry.Privilegios[0] == true) {%>
```

```
      <img class='avatar' src="http://localhost:3000/api/imagene
s/ImagenesAvatares/download/<%=entry.Complemento1%>">
      <% } else {%>
         <p style="font-family: Tahoma, Geneva, sans-
serif;">No puedes ver el complemento 1</p>
      <% }%>
      <% if (entry.Privilegios[1] == true) {%>
      <img class="avatar" src="http://localhost:3000/api/imagene
s/ImagenesAvatares/download/<%=entry.Complemento2%>">
      <% } else {%>
         <p style="font-family: Tahoma, Geneva, sans-
serif;">No puedes ver el complemento 2</p>
      <% }%>
      <% if (entry.Privilegios[2] == true) {%>
      <img class="avatar" src="http://localhost:3000/api/imagene
s/ImagenesAvatares/download/<%=entry.Complemento3%>">
      <% } else {%>
         <p style="font-family: Tahoma, Geneva, sans-
serif;">No puedes ver el complemento 3</p>
      <% }%>
      <% if (entry.Privilegios[3] == true) {%>
      <img class="avatar" src="http://localhost:3000/api/imagene
s/ImagenesAvatares/download/<%=entry.Complemento4%>">
      <% } else {%>
         <p style="font-family: Tahoma, Geneva, sans-
serif;">No puedes ver el complemento 4</p>
      <% }%>
    </br></br>
    <legend style="font-family: Tahoma, Geneva, sans-
serif;"><svg height="13" width="15"><circle cx="8" cy="7" r="6"
fill="brown" /></svg> Mi voz</legend>
    </br>
      <% if (entry.Privilegios[4] == true) {%>
        <% if (entry.Voz != '') {%>
        <audio src="http://localhost:3000/api/imagenes/ImagenesA
vatares/download/<%=entry.Voz%>" preload="none" controls></audio
>
        <% } else {%>
           <p style="font-family: Tahoma, Geneva, sans-
serif;">No tienes un fichero de Voz</p>
        <% }%>
      <% } else {%>
```

```
        <p style="font-family: Tahoma, Geneva, sans-
serif;">No puedes ver la nota de voz</p>
      <% }%>
    <% });%>
  </div>
</div>
    <div class="cover-container">
    <legend style="font-family: Tahoma, Geneva, sans-
serif;"><svg height="13" width="15"><circle cx="8" cy="7" r="6"
fill="yellow" /></svg> Mis privilegios</legend>
    </br>
    <table>
      <tbody>
        <thead>
          <tr>
          <%table.forEach(function(entry2) {%>
            <%for (let i = 0; i < 5; i++) {%>
              <%if (entry2.Privilegios[i] == true) {%>
                <th style="background-
color: #28c65a"><%=Object.keys(entry2)[i + 2]%></th>
              <% }%>
              <%if (entry2.Privilegios[i] == false) {%>
                <th style="background-
color: #ff0000"><%=Object.keys(entry2)[i + 2]%></th>
              <% }%>
            <% });%>
          </tr>
        </thead>
      </tbody>
    </table>
  </div>
</br></br>
  </body>
<div class="center">
  <button onclick="goBack()" class='buttonb buttonBack'>Go Back<
/button>
</html>
<script>
  function goBack() {
    window.history.back();
  }
```

```
  </script>
```

Lo único remarcable es la forma en la que se superponen las imágenes. Primero hacemos que la silueta del avatar sea la base (usando la clase avatarBase), tras eso, los complementes se añaden con posición absoluta (clase avatar) para poder ponerlos encima de la base:

```
  .avatarBase {
    position: relative;
  }

  .avatar {
    position: absolute;
    top: 45;
    left: 10;
  }
```

Dependiendo de si el alumno dispone de los privilegios necesarios o no, se mostrarán de una manera u otra:

```
<% if (entry.Privilegios[0] == true) {%>
      <img class='avatar' src="http://localhost:3000/api/imagene
s/ImagenesAvatares/download/<%=entry.Complemento1%>">
      <% } else {%>
         <p style="font-family: Tahoma, Geneva, sans-
serif;">No puedes ver el complemento 1</p>
      <% }%>
```

También añadí un apartado en el que visualizar la lista de privilegios del alumno:

```
    <div class="cover-container">
    <legend style="font-family: Tahoma, Geneva, sans-
serif;"><svg height="13" width="15"><circle cx="8" cy="7" r="6"
fill="yellow" /></svg> Mis privilegios</legend>
    </br>
    <table>
      <tbody>
        <thead>
          <tr>
          <%table.forEach(function(entry2) {%>
            <%for (let i = 0; i < 5; i++) {%>
```

```
              <%if (entry2.Privilegios[i] == true) {%>
                <th style="background-
color: #28c65a"><%=Object.keys(entry2)[i + 2]%></th>
              <% }%>
              <%if (entry2.Privilegios[i] == false) {%>
                <th style="background-
color: #ff0000"><%=Object.keys(entry2)[i + 2]%></th>
              <% }%>
            <% });%>
            </tr>
          </thead>
        </tbody>
      </table>
    </div>
```

Simplemente iteramos la tabla que se obtiene del servidor y, dependiendo de si el privilegio está activo (true) o no (false), lo marcamos de color verde o rojo respectivamente.

Para poder visualizar el avatar y la lista de privilegios lado a lado, se tiene que modificar el estilo del parámetro *div*:

```
<div class="cover-container">
```

En la sección de estilos, le añadimos algo de margen en el lateral derecho y listos:

```
.cover-container {
    display: inline-block;
    margin-right: 3%;
  }
```