Práctica Tema 1

Desarrollo de una API REST para el catálogo de una plataforma de juegos de mesa

En esta práctica vamos a desarrollar una serie de servicios REST para almacenar información del catálogo de juegos de una plataforma de juegos de mesa llamada **playREST**. Los pasos que seguiremos son los siguientes:

1. Información a almacenar

Debes crear un array de objetos Javascript que almacene la siguiente información para cada juego:

- **id,** código identificativo único. Puede ser un código numérico o alfanumérico, pero no puede haber dos juegos con el mismo código.
- **nombre del juego**
- **descripción del juego**
- **edad mínima recomendada en años**
- **número de jugadores**
- **tipo** (rol, cartas, dados, fichas, etc.)
- **precio**

2. Estructura de la aplicación

Crea un proyecto llamado **playREST** e instala el módulo o librería de Express (con el correspondiente comando `npm init` y `npm install`). Además, puedes emplear otras librerías adicionales del núcleo de Node.js, o de terceros, que necesites (por ejemplo, la librería `fs`, que te hará falta para almacenar datos en fichero).

La aplicación debe tener los siguientes archivos en la carpeta raíz:

- **`utilidades.js`**, donde estarán las funciones para cargar y guardar datos en formato JSON desde/en ficheros de texto.
- **`index.js`**, con el servidor principal Express y los servicios.

3. Las funciones de acceso a ficheros

En el archivo **`utilidades.js`** debes definir, al menos, las siguientes funciones:

- **`cargarJuegos`**, que recibe como parámetro un nombre de fichero y devuelve el array de objetos (juegos) Javascript que se hayan leído de él en formato JSON (se deberán convertir de JSON a Javascript). La función devolverá un array vacío si el fichero no existe o no pudo leerse.
- **`guardarJuegos`**, que recibe como parámetros un nombre de fichero y un array de objetos (juegos), y guardará los objetos del array en el fichero en formato JSON. Si

el array es nulo o vacío, no se hará nada con el fichero.

4. El servidor principal

El servidor principal debe incluir lo siguiente:

- Cargar la librería Express, y las que necesites.
- Incluir el fichero `utilidades.js`
- Definir el nombre de fichero donde cargar/guardar juegos y guardarlo en una constante. El fichero puede llamarse, por ejemplo, `juegos.json`.
- Cargar los juegos del fichero en un array Javascript antes de iniciar el servidor
- Definir el objeto Express con sus servicios, que se detallan a continuación.
- Poner en marcha el servidor Express en el puerto 8080

4.1. Servicios a desarrollar

Se pide desarrollar los siguientes servicios. Para cada uno, se detalla a qué URL debe responder. En TODOS los casos, se debe enviar:

- **Código de estado apropiado**: 200 si todo ha ido bien, 400 si hay un fallo en la petición y 500 si hay un fallo en el servidor.
- **Objeto JSON** con estos atributos:
  - **`error`**, que sólo estará presente si el código de estado es diferente a 200. Contendrá un mensaje con el error que se haya producido.
  - **`resultado`**, que sólo estará presente si ok es verdadero. Contendrá el resultado que se envía como respuesta. Dicho resultado se detalla a continuación para cada servicio.

Servicio GET /juegos

Atiende peticiones GET a la URI `/juegos`, devolviendo en el atributo resultado el array con todos los juegos.

Añade la posibilidad de recibir una query string con el parámetro **anyos** (/juegos?anyos=18), donde anyos es la edad mínima recomendada del juego a buscar. Filtra y devuelve solo los juegos que cumplan con dicha condición.

Si la edad indicada es un valor negativo, se devolverá un error de tipo 400 con el mensaje “Edad mínima recomendada en años inválida”.

En la query string también podremos recibir otro parámetro llamado **tipo** (/juegos?tipo=rol), donde tipo es el tipo de juego de los juegos a buscar (rol, cartas, dados, fichas, etc.).

Se pueden combinar ambos filtros (años y tipo). En el caso de un valor incorrecto en alguno de ellos, devuelve el mensaje de error del primero que falle.

Servicio GET /juegos/id

Atiende peticiones GET a la URI `/juegos/id`, donde id es el código identificativo del juego a buscar. En el atributo resultado se devolverá únicamente el objeto con el juego (sólo uno), y si no se encuentra, se devolverá un error de tipo 400 y el mensaje "Código de juego inexistente"

Servicio POST /juegos

Atiende peticiones POST a la URI `/juegos`, y recibirá en el cuerpo de la petición todos los datos de la misma. Si el id del nuevo juego no se encuentra en el array, se añadirá (usando la función push vista en los apuntes), y se devolverá en el atributo resultado el nuevo juego añadido. Si el id ya existe, se devolverá un código de estado 400 con el error "Código de juego repetido"

Servicio PUT /juegos/id

Atiende peticiones PUT a la URI `/juegos/id`, siendo id el código del juego a modificar. Se enviarán en el cuerpo de la petición los datos del juego (todos salvo el id, que no se permite cambiar), y se modificarán los datos del objeto en el array, devolviendo el objeto modificado como resultado. Si no se encuentra el juego con el id indicado, se enviará un código 400 con el mensaje "Juego no encontrado".

Servicio DELETE /juegos/id

Atiende peticiones DELETE a la URI `/juegos/id`, siendo el id el código del juego a borrar. Se enviará como resultado el objeto eliminado, y si no se encuentra, se enviará un código 400 con el mensaje "Juego no encontrado".

NOTA IMPORTANTE: al finalizar correctamente los servicios de POST, PUT o DELETE, se debe llamar a la función `guardarJuegos` para guardar los cambios que se hayan producido en el array.

4.2. Pruebas con Postman o Thunder Client

Se pide, además, elaborar una colección de pruebas Postman o Thunder Client llamada **playREST**, para probar cada uno de los servicios indicados en el apartado anterior. Expórtalos a un archivo con el mismo nombre.

5. Entrega y calificación

Debes compartirme tu repositorio (user: maycalle) y entregar un archivo ZIP en Aules y el enlace a tu github. Dentro, deberá contener:

- El proyecto **playREST** de Node, sin que contenga la carpeta `node_modules`.
- El archivo Postman exportado, con las pruebas de la colección

5.1. Calificación de la práctica

Los criterios para calificar la práctica son los siguientes:

- Estructura correcta del array de objetos Javascript (correcta definición de los atributos o campos de cada objeto, y de los objetos dentro del array): **0,25 puntos**
- Estructura correcta del proyecto, con información correctamente almacenada en el archivo package.json en cuanto a dependencias externas, y correcta ubicación de los ficheros fuente: **0,25 puntos**

- Archivo utilidades.js: **1,50 puntos**, repartidos así:
    - Función `cargarJuegos` con los parámetros y comportamiento adecuado: **0,60 puntos.**
    - Función `guardarJuegos` con los parámetros y comportamiento adecuado: **0,60 puntos**
    - Correcta exportación de las funciones para ser usadas en otros archivos: **0,30 puntos**
- Archivo principal index.js: **5,75 puntos** repartidos así:
    - Orden inicial correcto, con los `require/import`, constantes e inicialización de datos adecuado: **0,25 puntos**
    - Servicio `GET /juegos`: **1,50 puntos**
    - Servicio `GET /juegos/id`: **0,50 puntos**
    - Servicio `POST /juegos`: **1,25 puntos**
    - Servicio `PUT /juegos/id`: **1,25 puntos**
    - Servicio `DELETE /juegos/id`: **1,00 puntos**
- Colección Postman o Thunder Client, con los servicios correctamente añadidos para probarse: **1,75 puntos** (*0,25 puntos cada petición*)
- Claridad y limpieza del código, y uso de un comentario inicial en cada fichero fuente explicando qué hace: **0,5 puntos**

Penalizaciones a tener en cuenta

- Si algún servicio no devuelve los atributos con el nombre indicado, o no responde a la URI indicada, se calificará con 0 puntos, independientemente de lo bien o mal que esté el código.
    - ***Ejemplo 1***: si en el servicio GET `/juegos` enviamos en el objeto de respuesta un atributo result, en lugar de uno resultado, el servicio se calificará con un 0.
    - ***Ejemplo 2***: si en el servicio de borrado (DELETE) se responde a la URI `/juegos/borrar/id` en lugar de a `/juegos/id`, el servicio se calificará con un 0.
- La no eliminación de la carpeta `node_modules` en el archivo ZIP de entrega se penalizará con 1 punto menos de nota global de la práctica. **Esta penalización se verá incrementada en posteriores prácticas.**
- Si se sigue una estructura de proyecto, código y/o nombres de archivo distinta a la propuesta, y no se justifica debidamente, o dicha justificación no es satisfactoria, se penalizará la calificación global de la práctica con hasta el 50% de la nota de la misma.