Práctica Tema 2

Desarrollo de una API REST para el catálogo de una plataforma de juegos de mesa con Express y Mongoose

En esta práctica vamos a ampliar lo hecho en la práctica del tema anterior, para construir una API REST más compleja y completa sobre la aplicación de creación de una plataforma de juegos de mesa, pero esta vez utilizando una base de datos MongoDB y la API que ofrece la librería de Mongoose.

1. Estructura de la aplicación

Crearemos una aplicación llamada **playREST_V2**, e instalaremos dentro los módulos de *express* y *mongoose*. La aplicación estará estructurada en las siguientes carpetas y archivos (además del archivo `package.json` que generaremos con `npm init`):

- Archivo `index.js` en la raíz del proyecto, donde irá el servidor principal.
- Carpeta `models` donde almacenaremos los esquemas y modelos de la aplicación. Concretamente tendrá un archivo: `juego.js`, que completaremos después.
- Carpeta `routes` con el enrutador para la colección de juegos de mesa. Contendrá el archivo `juegos.js`.

2. Definiendo esquemas y modelos

En la carpeta `models` almacenaremos la definición de esquemas y modelos de nuestra base de datos. En concreto será uno, con los campos que se indican a continuación (se deben respetar los nombres de los campos, que se marcan en negrita, incluyendo mayúsculas o minúsculas si procede):

Modelo *juego*

En el archivo `models/juego.js` definiremos el esquema para la colección de juegos de mesa. Cada juego tendrá los siguientes campos:

- El **nombre** del juego (obligatorio, de tipo texto, con una longitud mínima de 6 caracteres).
- La **descripcion** del juego (texto largo obligatorio, sin tamaño mínimo).
- La **edad** mínima recomendada en años (numérico obligatorio, mayor que 0 y menor que 100)
- El número de **jugadores** permitido (numérico obligatorio)
- El **tipo** de juego (tipo enum con las siguientes categorías: rol, escape, dados, fichas, cartas y tablero)
- El **precio** del producto (numérico obligatorio, mayor que 0).
- La **imagen** del producto (texto con la ruta relativa, no obligatoria).
- **Ediciones** más modernas o temáticas, para jugar partidas diferentes pero sin perder la esencia del juego original. Por ejemplo, del Cluedo podemos encontrar diferentes versiones como la edición Juego de Tronos, la edición Big Bang Theory o la edición Junior. Este campo será un array de subdocumentos del tipo de esquema que comentaremos a continuación.

Para las ediciones, definiremos un esquema local a este modelo, que contendrá estos campos:

- El nombre de la **edicion** del juego. Será de tipo texto obligatorio.
- El **anyo** de lanzamiento de la edición. Será de tipo numérico (mayor que 2000 y menor que el año actual).

Recuerda exportar el modelo.

3. Los enrutadores

En la carpeta `routes` definiremos el enrutador del modelo anterior. En TODOS los servicios, se debe enviar:

- Un 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
- Un objeto JSON con estos atributos:
  - `ok`: de tipo booleano indicando si la petición se ha procesado satisfactoriamente o no
  - `error`: sólo estará presente si `ok` es falso. Contendrá el mensaje con el error que se haya producido
  - `resultado`: sólo estará presente si `ok` es verdadero. Contendrá el resultado que se envía como respuesta. Dicho resultado se detalla a continuación para cada servicio.

Enrutador para *juegos*

Este enrutador responderá a URIs que comiencen por `/juegos`. Se pide implementar estos servicios:

- `GET /juegos`: devolverá como `resultado` todo el listado de juegos, con todos sus campos, incluyendo la información de las ediciones. Si no hubiera juegos, se devolverá un código 500 con el mensaje `"No se encontraron juegos de mesa"`.

- `GET /juegos/:id`: devolverá como `resultado` la ficha para un juego a partir de su *id*. Si no se encuentra, se devolverá un error 400 con el mensaje `"Juego no encontrado"`.

- `POST /juegos`: añadirá el juego que se reciba en la petición a la colección. En dicha petición se enviarán los campos básicos del juego (es decir, todo menos el *id* y el array de las ediciones). Se devolverá como `resultado` el juego insertado, o un código 400 con el mensaje `"Error insertando el juego"`, si ha habido algún error.

- `PUT /juegos/:id`: permitirá modificar los datos básicos del juego (todo salvo el *id* y el array de ediciones). De nuevo, se enviarán en la petición los campos básicos del juego, como en POST. Se devolverá como `resultado` el juego modificado, o un código 400 con el mensaje `"Error modificando el juego"`.

- `POST /juegos/ediciones/:idJuego`: permitirá modificar el array de ediciones del juego con el *id* indicado, añadiendo la nueva versión del juego que se envíe en la petición. Se enviarán en la petición los campos con el nombre de la edición y el año de lanzamiento, y se devolverá como resultado el

juego entero modificado, o un código 400 y el mensaje `"Error modificando las ediciones del juego"`, en caso de error.

- `DELETE /juegos/:id`: permitirá eliminar un juego, junto con sus ediciones, a partir de su *id*. Como `resultado` se devolverá el juego eliminado con todos sus campos, o un error 400 y el mensaje `"Error eliminando el juego"`.

- `DELETE /juegos/ediciones/:idJuego/:idEdicion`: permitirá eliminar la edición con el *id* de la edición indicada, para el juego con el *id* de juego indicado. Devolverá como `resultado` el juego en su estado actual, o un error 400 y el mensaje `"Error eliminado la edición del juego"`.

4. El servidor principal

El servidor principal deberá:

- Cargar las librerías necesarias (al menos, *express* y *mongoose*)
- Cargar los enrutadores de juegos
- Conectar a una base de datos "juegos" de MongoDB
- Inicializar Express, y aplicar el middleware de *express.json* para procesar datos en formato JSON, y asociar los enrutadores respectivamente a las rutas de `/juegos`.
- Poner en marcha el servidor Express por el puerto 8080.

5. Pruebas con Postman o Thunder Client

Se pide, además, que elaboréis una colección de pruebas Postman o THunder Client llamada **playREST_V2**, para probar cada uno de los servicios indicados anteriormente. Expórtala a un archivo con el mismo nombre.

6. Entrega y calificación

Deberéis entregar un archivo ZIP o similar, con vuestro nombre y el prefijo "PracT2". Por ejemplo, si os llamáis José Pérez, el archivo de entrega deberá ser `PracT2_Jose_Perez.zip`. Dentro, deberá contener:

- El proyecto **playREST_V2** de Node, sin que contenga la carpeta `node_modules`.
- El archivo de Postman o Thunder Client exportado, con las pruebas de la colección.

6.1. Calificación de la práctica

Los criterios para calificar esta práctica son los siguientes:

- Estructura correcta del proyecto, con las carpetas y nombres de archivos indicados en el enunciado, archivo `package.json` correctamente definido con las dependencias incorporadas: **0,75 puntos**
- Modelo de datos: **2,5 puntos**:
    - Esquema y modelo básico para los juegos (sin las ediciones): **1,5 punto**

    - Esquema local para las ediciones: **1 puntos**

- Enrutadores: **3,5 puntos**, repartidos así:

    - Servicios implementados: **0,5 puntos** cada servicio (7 en total)

- Funcionalidad del archivo principal `index.js`: **1 puntos**
- Colección Postman, con los servicios correctamente añadidos para probarse: **1,75 puntos** (*0,25 puntos* para 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 recibe o 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 su código

> **Ejemplo**: si en el servicio `POST /juegos` recibe un campo `nombreJuego` en lugar del original `nombre` que figura en el esquema, el servicio se calificará con un 0.

> **Ejemplo 2**: 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 3**: 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 1,5 puntos menos de nota global de la práctica.

- 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.

- En el apartado de calificación de los **enrutadores**, se deberán obtener **al menos 1,75 puntos** del total de 3,5 para considerar aprobada la práctica (además de llegar a la nota mínima exigible).