Práctica Tema 2

Desarrollo de una API REST para recetas de cocina 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 recetas de cocina, 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 **Recetas_V2**, e instalaremos dentro los módulos de *express*, *mongoose* y *body-parser*. 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. Internamente tendrá dos archivos: `receta.js` e `ingrediente.js`, que completaremos después.
- Carpeta `routes` con los enrutadores para las diferentes colecciones. Contendrá los archivos `recetas.js` e `ingredientes.js`.

2. Definiendo esquemas y modelos

En la carpeta `models` almacenaremos la definición de esquemas y modelos de nuestra base de datos. En concreto serán dos, 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 *ingrediente*

En el archivo `models/ingrediente.js` definiremos el esquema para la colección de ingredientes. Para cada ingrediente almacenaremos:

- Su **nombre** (obligatorio, de tipo texto, con longitud mínima de 2 caracteres). No puede haber dos ingredientes con el mismo nombre.
- Su **tipo** (obligatorio, de tipo enumeración que podrá tomar los valores "carne", "pescado", "frutas y verduras", "pastas y arroces" y "otros").

Recuerda definir el modelo asociado al esquema (para que se almacenen los datos en una colección llamada *ingredientes*), y exportarlo para poderlo utilizar en otros archivos.

Modelo *receta*

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

- El **titulo** de la receta (obligatorio, de tipo texto, con una longitud mínima de 5 caracteres).
- El número de **comensales** para los que se prepara la receta (numérico obligatorio, mayor que 0).
- El tiempo de **preparacion** en minutos (numérico obligatorio, con valores positivos (> 0)).
- El tiempo de **coccion** en minutos (numérico obligatorio, con valores positivos (>= 0))
- La **descripcion** de la receta (texto largo obligatorio, sin tamaño mínimo).
- La **imagen** del resultado final (texto, no obligatoria).
- Los **elementos** necesarios para elaborar la receta. Este campo será un array de subdocumentos del tipo de esquema que comentaremos a continuación.

Para los elementos necesarios de la receta, definiremos un esquema local a este modelo, que contendrá estos campos:

- El **ingrediente** asociado al elemento. Será una referencia al modelo de ingrediente del otro archivo, y será de tipo obligatorio.
- La **cantidad** de ingrediente necesaria. Será de tipo numérico obligatorio, con valor positivo (> 0).
- La **unidad** en que se mide la cantidad del ingrediente (por ejemplo, "gramos", "litros", "cucharadas"...). Será de tipo texto obligatorio, con tamaño mínimo de 5.

Recuerda definir el modelo para el esquema de la receta (no para los elementos de la misma) y exportarlo.

3. Los enrutadores

En la carpeta `routes` definiremos los enrutadores asociados a cada modelo. 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 *ingredientes*

Este enrutador responderá a URIs que comiencen por `/ingredientes`. En concreto se pide implementar estos servicios:

- `GET /ingredientes`: devolverá como `resultado` el listado de ingredientes, con todos sus campos. Si no hubiera ingredientes, se devolverá un error 500 y el mensaje `"No se encontraron ingredientes"`
- `GET /ingredientes/:id`: devolverá como `resultado` la ficha del ingrediente a partir de su *id*. Si no se encuentra, se devolverá un error 400 y el mensaje `"Ingrediente no encontrado"`.
- `POST /ingredientes`: añadirá el ingrediente que se recibe en la petición a la colección, siempre que su nombre no exista ya. Se recibirán en la petición todos los campos del ingrediente (salvo el *id*) con los nombres indicados en el esquema, y se devolverá como `resultado` el ingrediente recién insertado. En caso de error, se enviará un código 400 y el mensaje `"Error al insertar el ingrediente"`. Opcionalmente, se puede intentar mostrar algo más de la información del error producido (extraída del objeto de error generado).
- `DELETE /ingredientes/:id`: eliminará el ingrediente de la colección siempre que no se esté utilizando en ninguna receta. Se devolverá como resultado el ingrediente eliminado. En caso de que ya exista en alguna receta, se devolverá un error 400 con el mensaje `"No se puede eliminar el ingrediente porque pertenece a alguna receta"`. En cualquier otro tipo de error, se enviará un código 400 con el mensaje `"Error eliminando ingrediente"`.

Enrutador para *recetas*

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

- `GET /recetas`: devolverá como `resultado` todo el listado de recetas, con todos sus campos, mostrando información detallada de cada ingrediente (es decir, deberás utilizar `populate` para mostrar los datos de los ingredientes). Si no hubiera recetas, se devolverá un código 500 con el mensaje `"No se encontraron recetas"`.
- `GET /recetas/:id`: devolverá como `resultado` la ficha para una receta a partir de su *id*. Si no se encuentra, se devolverá un error 400 con el mensaje `"Receta no encontrada"`.
- `POST /recetas`: añadirá la receta que se reciba en la petición a la colección. En dicha petición se enviarán los campos básicos de la receta (es decir, todo menos el *id* y el array de elementos que la componen), con sus nombres originales según el esquema. Se

devolverá como `resultado` la receta insertada, o un código 400 con el mensaje `"Error insertando receta"`, si ha habido algún error.

- `PUT /recetas/:id`: permitirá modificar los datos básicos de la receta (todo salvo el *id* y los elementos). De nuevo, se enviarán en la petición los campos básicos de la receta, como en POST, respetando los nombres originales. Se devolverá como `resultado` la receta modificada, o un código 400 con el mensaje `"Error modificando receta"`.

- `POST /recetas/elementos/:id`: permitirá modificar el array de elementos que componen la receta con el *id* indicado, añadiendo el nuevo elemento que se envíe en la petición. Se enviarán en la petición los campos del nuevo elemento (respetando los nombres originales), y se devolverá como resultado la receta entera modificada, o un código 400 y el mensaje `"Error modificando elementos de la receta"`, en caso de error.

- `DELETE /recetas/:id`: permitirá eliminar una receta, junto con los elementos que la componen, a partir de su *id*. Como `resultado` se devolverá la receta eliminada con todos sus campos, o un error 400 y el mensaje `"Error eliminando receta"`.

- `DELETE /recetas/elementos/:idReceta/:idElemento`: permitirá eliminar el elemento con el *id* de ingrediente indicado, para la receta con el *id* de receta indicado. Devolverá como `resultado` la receta en su estado actual, o un error 400 y el mensaje `"Error eliminado elemento de la receta"`.

4. El servidor principal

El servidor principal deberá:

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

5. Pruebas con Postman

Se pide, además, que elaboréis una colección de pruebas Postman llamada **Recetas_V2**, para probar cada uno de los servicios indicados anteriormente (tanto los de ingredientes como los de recetas). 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 Juan García, el archivo de entrega deberá ser

`PracT2_Juan_Garcia.zip`. Dentro, deberá contener:

- El proyecto **Recetas_V2** de Node, sin que contenga la carpeta `node_modules`.
- El archivo de Postman 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: **0,25 puntos**
- Modelo de datos: **1,5 puntos**, repartidos así:
  - Esquema y modelo para los **ingredientes**: **0,5 puntos**
  - Esquemas y modelo para las **recetas**: **1 puntos**
- Enrutadores: **5,5 puntos**, repartidos así:
  - Servicios implementados: **0,5 puntos** cada servicio (4 servicios para ingredientes, 7 para recetas, 11 en total)
- Funcionalidad del archivo principal `index.js`: **0,75 puntos**
- Colección Postman, con los servicios correctamente añadidos para probarse: **1,5 puntos** (*0,5 puntos* para las peticiones sobre ingredientes, *1 punto* para las peticiones sobre recetas)
- 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 /ingredientes` esperamos recibir un campo `nombreIngrediente` en lugar del original `nombre` que figura en el esquema, el servicio se calificará con un 0.

> **Ejemplo 2**: si en el servicio `GET /recetas` 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 `/recetas/borrar/id` en lugar de a `/recetas/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. 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.

- En el apartado de calificación de los **enrutadores**, se deberán obtener **al menos 2 puntos** del total de 5,5 para considerar aprobada la práctica (además de llegar a la nota mínima exigible).