

Práctica Tema 1

Desarrollo de una API REST para una web de productos asturianos

En esta práctica vamos a desarrollar una serie de servicios REST para almacenar datos sobre **productos asturianos**. Los pasos que seguiremos serán estos...

1. Información a almacenar

Crearemos un array de objetos JavaScript que almacene la siguiente información para cada producto:

- **Id**, código identificativo del producto. Puede ser un código numérico o alfanumérico, como prefieras, pero no puede haber dos productos con el mismo código.
- **Nombre** del producto. Por ejemplo, "*Queso Afuega'l Pitu (Rojo)*".
- **Precio**, en euros (incluyendo 2 decimales y símbolo del euro).
- **Descripción** del producto, un texto más o menos largo que explique su origen y/o elaboración junto con algún consejo para su presentación o consumo. Por ejemplo, "*Queso sencillo de sabor suave perfecto para tomar como entrante o aperitivo acompañado de una mermelada de Higo o de arándanos. Elaborado artesanalmente con leche pasteurizada de vaca.*"

2. Estructura de la aplicación

Deberás crear un proyecto llamado **ProdAsturianos_V1** en el que instalarás 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 los datos en fichero).

La aplicación tendrá dos archivos en la carpeta raíz:

- `fichero_utils.js`: donde se tendrán funciones de utilidad 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 `fichero_utils.js` se deberán definir, al menos dos funciones:

- `cargarProductos`: recibirá como parámetro un nombre de fichero, y devolverá el array de objetos (productos) JavaScript que se lea 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.
- `guardarProductos`: recibirá como parámetros un nombre de fichero, y un array de objetos (productos) para guardar, y guardará en formato JSON los objetos del array en el fichero. Si el array es

nulo, no se hará nada con el fichero.

4. El servidor principal

El servidor principal deberá:

- Cargar la librería Express y las que se necesiten
- Incluir el archivo `fichero_utils.js`
- Definir el nombre de fichero donde cargar/guardar productos en una constante. El fichero puede llamarse, por ejemplo, `productos.json`.
- Cargar los productos 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 por 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 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.

Servicio GET /productos

Atenderá peticiones GET a la URI `/productos`, devolviendo en el atributo `resultado` el array de productos actual. Si no hubiera productos en el archivo, se devolverá un error de tipo 500 y el correspondiente mensaje de error `"No se encontraron productos"`.

Servicio GET /productos/id

Atenderá peticiones GET a la URI `/productos/id`, donde `id` será el código identificativo del producto a buscar. En el atributo `resultado` se devolverá únicamente el objeto con el producto encontrado (sólo uno), y si no se encuentra, se devolverá un error de tipo 400, y el mensaje `"Producto no encontrado"`.

Servicio POST /productos

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

Servicio PUT /productos/id

Atenderá peticiones PUT a la URI `/productos/id`, siendo *id* el código del producto a modificar. Se enviarán en el cuerpo de la petición los datos del producto (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 producto con el *id* indicado, se enviará un código 400 con el mensaje `"Producto no encontrado"`.

Servicio DELETE /productos/id

Atenderá peticiones DELETE a la URI `/productos/id`, siendo *id* el código del producto a borrar. Se enviará como `resultado` el objeto eliminado, y si no se encuentra, se enviará un código 400 con el mensaje `"Producto no encontrado"`.

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

4.2. Pruebas con Postman

Se pide, además, que elaboréis una colección de pruebas Postman llamada **ProdAsturianos_V1**, para probar cada uno de los servicios indicados en el apartado anterior. Expórtala a un archivo con el mismo nombre.

5. Entrega y calificación

Deberéis entregar un archivo ZIP o similar, con vuestro nombre y el prefijo "PracT1". Por ejemplo, si os llamáis Juan García, el archivo de entrega deberá ser `PracT1_Juan_Garcia.zip`. Dentro, deberá contener:

- El proyecto **ProdAsturianos_V1** de Node, sin que contenga la carpeta `node_modules`.
- El archivo de Postman exportado, con las pruebas de la colección.

5.1. Calificación de la práctica

Los criterios para calificar esta 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): **0,5 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: **0,5 puntos**
- Archivo `fichero_utils.js`: **1,75 puntos**, repartidos así:
 - Función `cargarProductos` con los parámetros y comportamiento adecuado: **0,75 puntos**
 - Función `guardarProductos` con los parámetros y comportamiento adecuado: **0,75 puntos**
 - Correcta exportación de las funciones para ser usadas en otros archivos: **0,25 puntos**
- Archivo principal `index.js`: **5,5 puntos**, repartidos así:

- Orden inicial correcto, con los `require`, constantes e inicialización de datos adecuada: **0,5 puntos**
 - Servicio `GET /productos` : **0,75 puntos**
 - Servicio `GET /productos/id` : **0,75 puntos**
 - Servicio `POST /productos` : **1,25 puntos**
 - Servicio `PUT /productos/id` : **1,25 puntos**
 - Servicio `DELETE /productos/id` : **1 punto**
- Colección Postman, con los servicios correctamente añadidos para probarse: **1,25 punto** (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 su código

Ejemplo: si en el servicio `GET /productos` 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 `/productos/borrar/id` en lugar de a `/productos/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.