Práctica 3: Backend

Crear un servicio web RESTful

Objetivo: Crear el **backend** de la aplicación de **e-commerce** manejada hasta el momento.

Descripción de la práctica

Usando **Node.js** se creará una **REST API** que permitirá crear, modificar, editar, consultar y borrar productos. Para facilitar las cosas, la información se guardará en archivos **JSON** y se realizará una autenticación *básica*. Para validar el servicio web, únicamente se hará uso de solicitudes HTTP para realizar operaciones de altas, bajas, cambios y consultas usando **Postman**.

Los contenidos de la práctica son los siguientes:

1. Configuración inicial
2. Estructura de archivos sugerida
3. Acciones, rutas y métodos
4. Comportamiento de rutas
    a. GET /products
    b. POST /products/cart
    c. GET /products/:id y middleware de autenticación
    d. POST /admin/products
    e. PUT /admin/products/:id
    f. DELETE /admin/products/:id
    g. GET /, /home, /shopping_cart
5. Entregables
6. Evaluación

Configuración inicial

- Crea una carpeta que se llame *practica3*
- En la consola, ejecuta **`npm init –yes`**
- Instala como dependencia ***express***
- Crea un archivo **products.json** (ahí se guardará la información de los productos); Inicialmente tendrá un arreglo vacío **[]**
- Crea un archivo **server.js** y **data_handler.js**
    - Importa ***express***
    - Carga de forma global el middleware que permite parsear el archivo JSON (***express.json()***)
    - Crea una ruta raíz que solo regresa “e-commerce app práctica 3”
- Crea un archivo ***data_handler.js***
    - Importa ***fs***
    - En una variable **products** carga y parsea el archivo **products.json**
- Levanta el servicio en el puerto 3000

Comprueba que todas estas configuraciones funcionen correctamente.

Estructura de archivos sugerida

Se recomienda la siguiente estructura para la práctica (observa que algunos archivos corresponden a prácticas pasadas).

```
practica3/
└── app
    ├── controllers
    │   ├── data_handler.js
    │   ├── product.js
    │   ├── router.js
    │   ├── shopping_cart.js
    │   └── utils.js
    ├── data
    │   └── products.json
    ├── routes
    │   ├── admin_products.js
    │   └── products.js
    └── views
        ├── home.html
        └── shopping_cart.html
```

Acciones, rutas y métodos

El servicio es capaz de realizar las siguientes acciones:

1. Consultar productos
2. Consultar producto en específico
3. Consultar productos según una lista de IDs
4. Registro de nuevos productos (con validaciones)
5. Editar información de un producto
6. Borrar algún producto
7. Filtro de productos (obtener todos o algunos según los parámetros)
8. Manejo de errores de respuesta de HTTP
9. Validación de headers usando middlewares

Endpoint	Método	Headers	Body	Comentarios
/products/	GET			Si se pasa el parámetro "query" aplicar el filtro correspondiente.
/products/:id	GET			Regresar el producto correspondiente; status 200. Si el ID no coincide, mandar status 404.
/products/cart	POST	Content-Type: application/json	Arreglo de objetos con los atributos: productUuid, amount	Regresar un arreglo con todos los productos correspondientes; status 200. Si el ***body*** no es un arreglo, mandar status 400. Si algún ID no coincide, mandar status 404.
/admin/products/	POST	X-Auth Content-Type: application/json	uuid, imageUrl, title, description, unit, category, pricePerUnit, stock	Crear el producto correspondiente. En caso de éxito, mandar status 201, si no, mandar status 400.
/admin/products/:id	PUT	X-Auth Content-Type: application/json	imageUrl, title, description, unit, category, pricePerUnit, stock	Actualizar el producto correspondiente. En caso de éxito, mandar status 200, si no, mandar status 400. Si el ID no coincide, mandar status 404.
/admin/products/:id	DELETE	X-Auth		Borrar el producto correspondiente; en caso de éxito, mandar status 200. Si el ID no coincide, mandar status 404.

GET /products

Lo primero a realizar será el retorno de la lista de productos, para lo cual se usarán como apoyo los archivos de la *práctica 2*. En caso de haber implementado la parte opcional de filtros, es posible agregarla aquí para retornar la lista filtrada.

1. Regresar todos los productos: se deben regresar los valores que ya estén cargados en el arreglo de productos (**products**).
2. Regresar lista filtrada de productos: verificar si la petición cuenta con *parámetros de búsqueda*, en cuyo caso se mandará la lista filtrada según los parámetros pasados (consultar la *práctica 2* para ver cuáles son).

POST /products/cart

Para poder cargar los productos correctamente en el carrito de compras, se debe permitir la búsqueda de productos *directamente por su **ID***.

1. Se tendrá que validar que el *body* recibido sea de tipo *arreglo*, de lo contrario, se debe regresar **status de error 400**.
2. Se iterará sobre el arreglo recibido y se irán agregando productos a una lista temporal. En caso de que no se encuentre algún producto, se debe regresar el status de error 404 y un mensaje explicando que no se encontró dicho producto.
3. Al terminar, si y solo si todos los productos fueron encontrados, se regresará la lista temporal de productos con un **status de éxito 200**. Recuerda agregar a la respuesta el encabezado (header) correspondiente para JSON.

GET /products/:id

Se debe permitir la búsqueda de productos directamente por su ***ID***.

1. En caso de que el ID sea inválido, se debe regresar **status de error 404** con un mensaje explicando que no se encontró el producto.
2. En caso de que todo sea correcto, se debe regresar **status de éxito 200** y el producto correspondiente. Recuerda agregar a la respuesta el encabezado correspondiente para JSON.

Middleware de autenticación

Se debe crear un middleware llamado **validateAdmin** que verifica que exista el header ***X-Auth*** con el valor **"admin"**. En caso de que no exista, se debe regresar **status de error 403** con el mensaje "Acceso no autorizado, no se cuenta con privilegios de administrador".

POST /admin/products

Para la parte de administrador, lo primero que se debe realizar será el manejo de registro de un nuevo producto. Antes de continuar, *no olvides poner el* ***middleware de autenticación***. Para hacer el registro, se deben seguir los siguientes pasos:

1. Validar que dentro del *body* se encuentren los atributos relevantes para la creación del producto. Si los atributos son incorrectos, regresar el **status de error 400** e indicar en un mensaje cuáles atributos faltan (si sobran, se pueden ignorar internamente). Es posible usar **try/catch** en conjunto de las validaciones de la *práctica 2*.
2. Se debe asegurar que el ***ID*** del nuevo producto se autogenere usando UUIDs
    a. Utilizar ***generateUUID*** o instalar el paquete ***uuid*** (buscar cómo se utiliza)
3. Guardar el nuevo producto en el arreglo y regresar **status de éxito 201** y como mensaje el nombre del producto creado.
4. Después de esto, actualizar el archivo de productos.

PUT /admin/products/:id

Se debe permitir al administrador modificar los atributos de algún producto. *Aquí también debe incluirse el* ***middleware de autenticación***.

1. Validar que el ***ID*** corresponda a un producto existente, de lo contrario, regresar el **status de error 404**.
2. Validar que en el *body* se encuentren los atributos relevantes (similar al **POST**).
3. Guardar los cambios del producto en el arreglo (en su posición correspondiente, no como nuevo elemento) y regresar **status de éxito 200** con un mensaje indicando el nombre del producto que fue actualizado.
4. Después de esto, también se debe actualizar el archivo de productos.

DELETE /admin/products/:id

Finalmente, el administrador también debe poder borrar un producto en específico. Nuevamente, *se debe incluir el* ***middleware de autenticación***.

1. Validar que el ***ID*** corresponda a un producto existente, de lo contrario, regresar el **status de error 404**.
2. Eliminar el producto del arreglo y regresar **status de éxito 200** con un mensaje indicando el nombre del producto que fue borrado.
3. Nuevamente, actualizar el archivo de productos después de la operación.

GET /, /home, /shopping_cart

Investiga cómo mostrar los archivos HTML de la práctica 1, de forma que:

1. Al ingresar a la ruta raíz (**/**) o a la ruta de home (**/home**) se muestre **home.html**
2. Al ingresar a la ruta del carrito de compras (**/shopping_cart**) se muestre **shopping_cart.html**

Entregables

1. Archivo **.zip** (no .rar ni otros formatos) con los archivos solicitados
    a. Dentro debe ir una colección (formato JSON) de **Postman** ó un archivo **HTTP** de REST Client, validando el funcionamiento de cada Endpoint
2. **Vídeo *narrado*** demostrando la práctica y todas las funcionalidades solicitadas en las secciones anteriores; se recomienda que te apoyes de este documento para demostrar cada parte de la misma. ***Sin video, no hay calificación.***
    a. El vídeo ***no debe ir dentro del zip***

Evaluación

Requerimiento (incluye validaciones, regresar código de error y que funcione correctamente)	Puntuación
1. GET /products - Filtro por parámetro de búsqueda	**15**
2. POST /products/cart	**15**
3. GET /products/:id	**10**
4. Middleware de autenticación	**10**
5. POST /admin/products	**10**
6. PUT /admin/products/:id	**15**
7. DELETE /admin/products:id	**10**
8. GET /, /home, /shopping_cart	**10**
9. Código estructurado y modularizado - Archivo de router global - Archivo de router para productos - Archivo de router para endpoints de administrador de productos	**10**
10. Manejo de archivos	**10**

Máxima calificación de la práctica: 100 puntos.

Consideraciones adicionales

- El alcance de la práctica llega hasta lo descrito únicamente, por ende, si se desean realizar implementaciones adicionales y no funcionan, habrá una penalización por ello
- El vídeo debe ser conciso; puedes explayarte tanto como desees, siempre y cuando lo que expliques sea relevante para la práctica. Procura no producir algo extenso
- Si no estás seguro de algo o te parece ambigua alguna instrucción, ***PREGUNTA***
    - No hagas asunciones, es posible que tu entendimiento de algún punto sea distinto a lo que se pide