Guía 2. Creación del API RESTFul Básico

En esta ocasión vamos a crear un sencillo Servicio Web que proporciona una **API RESTful** basado en **HTTP** (comandos **GET**, **POST**, **PUT** y **DELETE**). Este servicio no tendrá lógica de negocio, por lo que realmente no hará nada. Sin embargo, nos servirá para entender cómo se puede desarrollar y cómo funciona un servicio web con tecnología REST.

Es fundamental entender cómo funciona la comunicación entre el cliente y el servicio sobre HTTP, por lo que el estudiante deberá ampliar esta guía con un pequeño estudio que debe incluir como mínimo el patrón de intercambio basado en petición-respuesta, el formato de una petición HTTP (*HTTP Request*) y de una respuesta HTTP (*HTTP Request*), y las características básicas de este tipo de comunicación.

El contenido del resto de la guía se estructura de la siguiente manera:

1. Preparando utilidades básicas
2. Creamos un API REST CRUD básico sin base de datos
3. Analizamos las herramientas incorporadas

Preparando utilidades básicas

Antes de empezar, instalaremos una serie de herramientas, tanto externas a la pila MEAN (**JSON formatter** y **Postman**), como de la pila MEAN (**Nodemon** y **Morgan**) que nos facilitarán enormemente el trabajo en el futuro.

JSON Formatter

Es muy interesante instalar en nuestro navegador algún módulo que interprete **JSON** (un **JSON Formatter**) para tener una mejor visualización de las respuestas de nuestro API. Por ejemplo https://github.com/rfletcher/safari-json-formatter, descargar el paquete y abrirlo con Safari. Aunque es posible que vuestro navegador ya tenga un formateador de JSON incorporado y os resulte suficiente. Si es así, este paso se puede obviar.

Postman

Igualmente, instalamos **Postman** (o similar) en el navegador o en su versión de escritorio (https://www.getpostman.com/). para poder hacer invocaciones (HTTP Request) a nuestro servidor diferentes de **GET** (como **PUT**, **DELETE**, etc.).

En Ubuntu se puede descargar e instalar directamente desde su almacén de aplicaciones.

Nodemon

Es un gestor de proyectos que, entre otras cosas, impide que tengamos que estar constantemente reiniciando nuestra aplicación con cada cambio en el código fuente.

Para su instalación, en primer lugar, abrimos una terminal y nos dirigimos a la carpeta en la que se encuentra nuestro proyecto:

```
<Ctrl+Alt+T>
$ cd
$ cd node
$ cd api-rest
```

A continuación, instalamos la biblioteca **nodemon** mediante la siguiente orden:

```
$ npm i -D nodemon #equivale a: npm install —devDependencies nodemon
```

Nota: en este caso, se emplea la directiva -D, en lugar de -S. Esto se debe a que **nodemon** es una herramienta que nos será útil sólo durante el desarrollo, pero no en producción. Por esta razón, se registrará una entrada en el **package.json pero en la sección devDependencies**.

```
"devDependencies": {
        "nodemon": "^1.17.2"
}
```

Podemos iniciar nuestro editor para seguir trabajando con nuestra aplicación.

```
$ code .
```

Utilizando nuestro editor incluimos una línea (resaltada en negrita: **"start": "nodemon index.js",**) en la sección scripts del archivo **package.json** para crear un script de inicio que invoque **nodemon**

```
{
        "name": "api-rest",
        "version": "1.0.0",
        "description": "Proyecto de API RESTful con Node.js y Express para SD",
        "main": "index.js",
        "scripts": {
                "start": "nodemon index.js",
                "test": "echo \"Error: no test specified\" && exit 1"
        },
        "keywords": [
                "RESTfull",
                "node",
                "express",
                "mongo"
        ],
        "author": "Paco Maciá",
        "license": "ISC",
        "dependencies": {
                "express": "^4.16.3"
        },
        "devDependencies": {
                "nodemon": "^1.17.2"
        }
}
```

Ahora podemos reiniciar el proyecto de forma que cada vez que hagamos alguna modificación en alguno de los archivos implicados, automáticamente se actualizará el ejecutable. Por lo tanto, a partir de ahora, en lugar del tradicional comando **node index.js** que veníamos ejecutando, lo haremos de la siguiente forma:

```
$ npm start
> api-rest@1.0.0 start $HOME/node/RESTFulServer
> nodemon index.js
[nodemon] 1.17.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node index.js`
API REST ejecutándose en http://localhost:3000/hola/:unNombre
```

Ahora incorporaremos algún cambio en nuestro ejemplo para verificar que realmente **nodemon** está haciendo su trabajo. Concretamente, cambiaremos el puerto de escucha por el puerto **3000**.

```
'use strict'

const port = process.env.PORT || 3000;

const express = require('express');
```

```
const app = express();

app.get('/hola/:name', (req, res) => {
    res.status(200).send({mensaje: `¡Hola ${req.params.name} desde SD con JSON!`});
});

app.listen(port, () => {
    console.log((`API REST ejecutándose en http://localhost:${port}/hola/:unNombre`);
});
```

Gracias a **nodemon**, ahora ya no tendremos que detener el anterior servidor, y volver a lanzar la nueva versión. Cada vez que hagamos un cambio en algunos de los archivos del proyecto, podremos observar a través de la terminal de texto que el servicio se reinicia automáticamente:

```
API REST ejecutándose en http://localhost:3000/hola/:unNombre
```

Lo probamos de nuevo con nuestro navegador conectándonos a la URL: http://localhost:3000/hola/Estudiantes) y el resultado debe ser:

```
{"mensaje":"¡Hola Estudiantes desde SD con JSON!"}
```

Aunque sí nuestro navegador soportaba el formato JSON o hemos instalado correctamente nuestro formateador, podremos ver algo como:

```
mensaje: ¡Hola Estudiantes desde SD con JSON!
```

Morgan

Cuando estamos desarrollando, es muy importante saber qué está pasando en cada momento en nuestro servidor. Para ello, debemos tener una política de registro (*logs*). En este caso, como estamos buscando una versión sencilla de todo lo que hacemos, hemos seleccionado **Morgan** como motor de registro. **Morgan** no es tan versátil o poderosa como sus competidoras, pero proporciona registro de peticiones y respuestas en aplicaciones **Express**. Además, está escrito como *middleware* de **NodeJS**. Para instalarlo, siguiendo el método visto hasta el momento, bastará con ejecutar la siguiente instrucción:

```
$ npm i -S morgan
```

Ahora incorporaremos el **logger**: primero lo importamos y después lo configuramos como un **middleware** de **Node** de forma que actuará automáticamente.

```
'use strict'

const port = process.env.PORT || 3000;

const express = require('express');
const logger = require('morgan');

const app = express();

// Declaramos los middleware
app.use(logger('dev'));    // probar con: tiny, short, dev, common, combined

// Declaramos el API
app.get('/hola/:name', (req, res) => {
    res.status(200).send({mensaje: `¡Hola ${req.params.name} desde SD con JSON!`});
});

app.listen(port, () => {
```

```
    console.log((`API REST ejecutándose en http://localhost:${port}/hola/:unNombre`);
});
```

Si nos fijamos en la terminal, ahora tendremos más información acerca de lo que está ocurriendo gracias a **Morgan**. A continuación, mostramos una salida en la que se ha realizado correctamente un **GET**:

```
$ npm start
API REST ejecutándose en http://localhost:3000/api/product
GET /hola/Juan 200 0.452 ms - 41
^C
```

Antes de seguir, guardaremos en el repositorio remoto los cambios realizados. Como la terminal está ocupada con nuestro servicio (que ahora no se detiene nunca gracias a **Nodemon**), tendremos que dormirlo, detenerlo o, sencillamente, abrir una nueva terminal para tareas de gestión, como la sincronización del repositorio, la instalación de bibliotecas, etc.:

```
$ git add .
$ git commit -m "Primer API REST con Express y JSON"
$ git push
$ git status
```

Primer API REST tipo CRUD

Podemos escribir ya un pequeño servicio (una pequeña aplicación) que atienda los principales métodos **HTTP** que darán soporte a nuestra interfaz **RESTful** (**GET**, **POST**, **PUT**, **DELETE**).

Aunque sería mucho mejor pensar y especificar primero qué interfaz, que API deseamos para nuestro servicio, por el momento lo haremos de forma constructiva comenzando por el código.

```
'use strict'

const port = process.env.PORT || 3000;
const express = require('express');
const logger = require('morgan');

const app = express();

// Declaramos los middleware
app.use(logger('dev'));    // probar con: tiny, short, dev, common, combined

// Implementamos el API RESTFul a través de los métodos
app.get('/api/products', (req, res) => {
      res.status(200).send({products: []});
});

app.get('/api/products/:productID', (req, res) => {
      res.status(200).send({products: `${req.params.productID}`});
});

app.post('/api/products', (req, res) => {
      console.log(req.body);
      res.status(200).send({products: 'El producto se ha recibido'});
});

app.put('/api/products/:productID', (req, res) => {
      res.status(200).send({products: `${req.params.productID}`});
});

app.delete('/api/products/:productID', (req, res) => {
      res.status(200).send({products: `${req.params.productID}`});
});
```

```
// Lanzamos nuestro servicio API
app.listen(port, () => {
	console.log(`API REST ejecutándose en http://localhost:${port}/api/product`);
});
```

Probamos GET (sin parámetros) introduciendo la URL en el navegador http://localhost:3000/api/product

Resultado:

```
products: []
```

Probamos un **POST** con **Postman** a la misma URL. Para ello seleccionaremos el método **POST** y enviamos los parámetros con la opción del body a **x-www-form-urlencoded** (en modo key-value edit). Esto es equivalente a añadir en la cabecera el campo:

```
Content-Type: application/x-www-form-urlencoded
```

Y en el **body** escribimos, por ejemplo:

```
name		mi producto
price		200
photo		miProducto.png
category	general
```

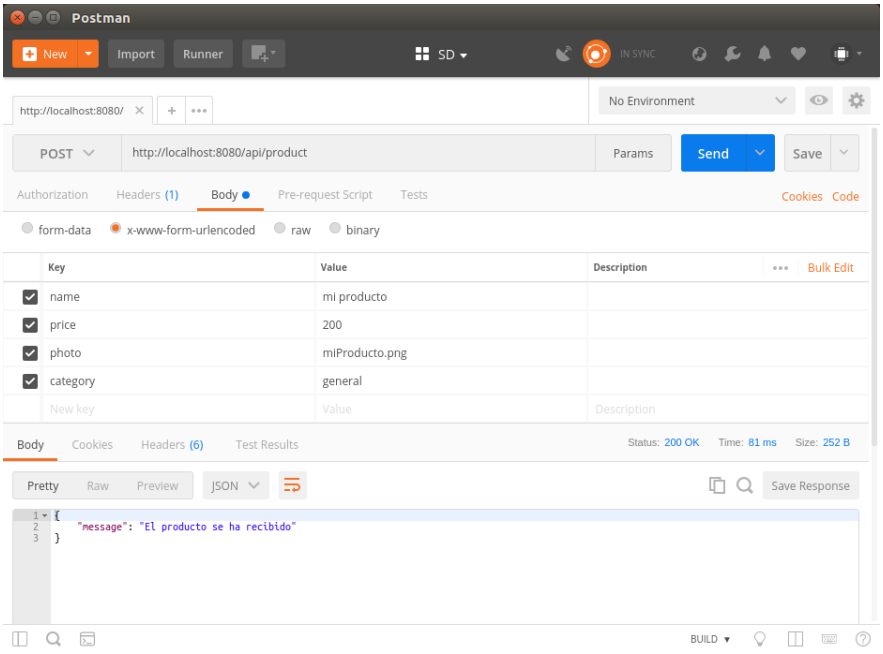
Resultado en **Postman**:

```
código 200 OK
{
	"message": "El producto se ha recibido"
}
```

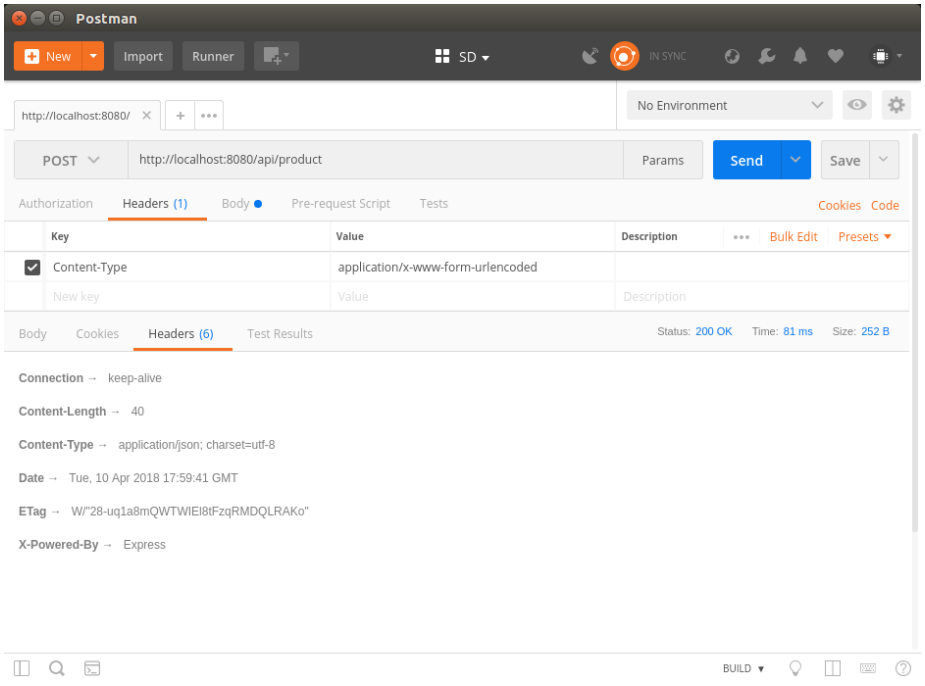
La cabecera devuelta será algo similar a:

```
Connection: keep-alive
Content-Length: 40
Content-Type: application/json; charset=utf-8
Date: Tue, 10 Apr 2019 17:59:41 GMT
ETag: W/"28-uq1a8mQWTWIEl8tFzqRMDQLRAKo"
X-Powered-By: Express
```

En la siguiente figura podemos ver una captura de pantalla de **Postman** tras ejecutar el ejemplo propuesto. Tanto para la solicitud como para la respuesta se está mostrando la pestaña **<Body>**.



En la siguiente figura podemos ver otra captura de pantalla de **Postman** tras ejecutar el ejemplo **POST** propuesto en la que se ha seleccionado las pestañas **<Header>** tanto para la solicitud como para la respuesta.



Si nos fijamos en la terminal, ahora tendremos más información acerca de lo que está ocurriendo gracias a **Morgan**: a continuación, mostramos una salida en la que se ha realizado correctamente un **GET**, un **POST** y un **PUT**. Así mismo, hemos tenido problemas con el último **PUT**:

```
$ npm start
API REST ejecutándose en http://localhost:3000/api/product
GET /api/product/ 200 0.452 ms - 41
undefined
POST /api/product/ 200 3.483 ms - 41
PUT /api/product/22 200 0.516 ms - 41
```

```
PUT /api/product/ 404 0.914 ms - 151
^C
```

En el caso del método **POST**, como además del middleware **Morgan**, también hemos utilizado `console.log`, para saber qué se le está pasando en el **body** de la petición, en la terminal habríamos esperado que se nos mostrase algo del tipo:

```
API REST ejecutándose en http://localhost:3000/api/product
{ name: 'mi producto',
  price: '200',
  photo: 'miProducto.png',
  category: 'general' }
POST /api/product/ 200 3.483 ms - 86
```

Que se corresponde con lo que habíamos enviado en el **body** de nuestra solicitud **POST**. Sin embargo, si nos fijamos en la terminal, en realidad lo que vemos es algo como:

```
$ npm start
API REST ejecutándose en http://localhost:3000/api/product
undefined
POST /api/product/ 200 3.483 ms - 41
^C
```

Este `undefined` se debe a que nuestra aplicación no sabe interpretar la información que le estamos enviando en el **body** de la petición (del HTTPRequest). Para resolver este problema configuraremos dos nuevos *middlewares*:

```
// Declaramos los middleware
app.use(express.urlencoded({ extended: false }));
app.use(express.json());
```

El primer *middleware* nos permitirá entender un **body** típico generado por un formulario web estándar o mediante un **body** generado con **Postman**, como es el caso del ejemplo anterior. En cualquier caso, este middleware actuará siempre y cuando en la cabecera de la petición HTTP se envíe la directiva:

`"Content-Type: application/x-www-form-urlencoded".`

De hecho, cuando seleccionemos este formato desde **Postman**, podremos comprobar que nos incluye automáticamente en la cabecera de la petición esta directiva.

El segundo middleware permite que nuestra aplicación entienda un **body** que contenga un objeto **JSON**. Cuando le decimos a **Postman** que el **body** tiene este formato, nos incluirá automáticamente en la cabecera la directiva:

`"Content-Type: application/json"`

Será justamente la directiva `Content-Type` la que determine qué middleware se debe utilizar en cada caso para interpretar qué es lo que enviamos en el **body** de la petición.

Se debe tener en cuenta que, aunque se le diga mediante `Content-Type` que el **body** es de un determinado formato, no tiene por qué serlo. Por ejemplo, si le indicamos que es un objeto **JSON**, pero luego no enviamos un objeto de estas características o está mal formado, obtendremos un error bastante aparatoso en nuestra terminal.

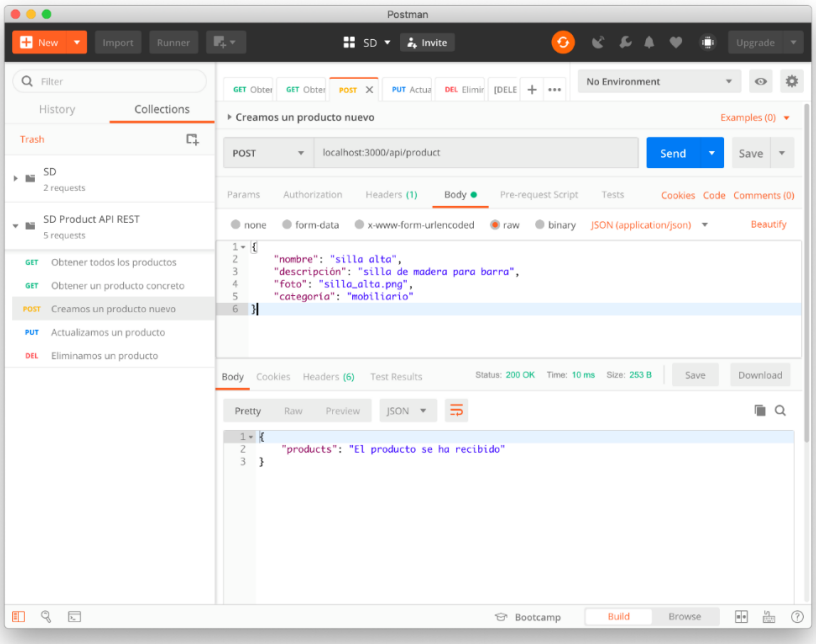
A continuación, podemos observar la salida que obtendremos por la terminal si incorporamos los dos *middlewares* mencionados. Concretamente la salida se corresponde a una prueba de un **POST** en el que se ha enviado el **body** codificado en formato `application/x-www-form-urlencoded`, otro **POST** con el formato `application/json`, pero enviando un objeto que no es **JSON** y, finalmente, otro **POST** con el mismo formato `application/json` y un objeto **JSON** bien formado:

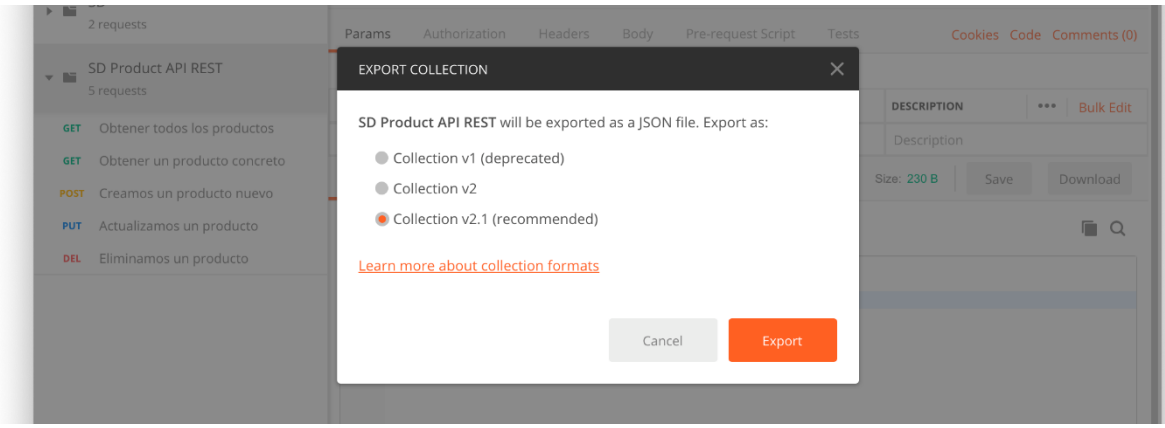
```
$ npm start
API REST ejecutándose en http://localhost:3000/api/product

[Object: null prototype] {
  prueba: 'x-www-form-urlencoded',
  middleware: 'urlencoded',
  varibale: 'valor' }
POST /api/product/ 200 10.078 ms - 41

SyntaxError: Unexpected token m in JSON at position 0
    at JSON.parse (<anonymous>)
    at createStrictSyntaxError (/Users/pmacia/OneDrive/
Prácticas/RESTFulServer/node_modules/body-parser/lib/types/json.js:158:10)
    at parse (/Users/pmacia/OneDrive/Prácticas/RESTFulServer/node_modules/body-
parser/lib/types/json.js:83:15)
POST /api/product/ 400 3.483 ms - 1116

{ name: 'mi producto',
  price: '200',
  photo: 'miProducto.png',
  category: 'general' }
POST /api/product/ 200 3.483 ms - 41
^C
```





El siguiente listado recoge la definición exportada desde **Postman** con una batería de llamadas al API REST de prueba, incluyendo consulta, creación, actualización y eliminación de elementos dentro de colecciones.

```
{
   "info": {
      "_postman_id": "4426a814-5e34-4477-bed8-ddc71c91e7a6",
      "name": "SD Product API REST",
      "description": "Paquete de prueba para validar la API RESTFul de simulación de productos de SD",
      "schema": "https://schema.getpostman.com/json/collection/v2.1.0/collection.json"
   },
   "item": [
      {
         "name": "Obtener todos los productos",
         "request": {
            "method": "GET",
            "header": [],
            "body": {
               "mode": "raw",
               "raw": ""
            },
            "url": {
               "raw": "http://localhost:3000/api/product",
               "protocol": "http",
               "host": [
                  "localhost"
               ],
               "port": "3000",
               "path": [
                  "api",
                  "product"
               ]
            },
            "description": "Verificamos el método GET para obtener todos los productos existentes"
         },
         "response": []
      },
      {
         "name": "Obtener un producto concreto",
         "request": {
            "method": "GET",
            "header": [],
            "body": {
               "mode": "raw",
               "raw": ""
            },
            "url": {
               "raw": "localhost:3000/api/product/258",
               "host": [
                  "localhost"
               ],
               "port": "3000",
               "path": [
                  "api",
                  "product",
                  "258"
               ]
            },
            "description": "Prueba del API REST para obtener un único producto"
         },
         "response": []
      },
      {
         "name": "Creamos un producto nuevo",
         "request": {
            "method": "POST",
            "header": [
               {
                  "key": "Content-Type",
                  "name": "Content-Type",
                  "value": "application/json",
                  "type": "text"
               }
```

```
            ],
            "body": {
                "mode": "raw",
                "raw": "{\n\t\"nombre\": \"silla alta\",\n\t\"descripción\": \"silla de madera para
barra\",\n\t\"foto\": \"silla_alta.png\",\n\t\"categoría\": \"mobiliario\"\n}"
            },
            "url": {
                "raw": "localhost:3000/api/product",
                "host": [
                    "localhost"
                ],
                "port": "3000",
                "path": [
                    "api",
                    "product"
                ]
            },
            "description": "Validación del API REST productos para crear productos (simuladamente)"
        },
        "response": []
    },
    {
        "name": "Actualizamos un producto",
        "request": {
            "method": "PUT",
            "header": [
                {
                    "key": "Content-Type",
                    "name": "Content-Type",
                    "value": "application/json",
                    "type": "text"
                }
            ],
            "body": {
                "mode": "raw",
                "raw": "{\n\t\"nombre\": \"silla baja\",\n\t\"descripción\": \"silla de madera para
mesa\",\n\t\"foto\": \"silla_baja.png\",\n\t\"nuevo campo\": \"interesante\"\n}"
            },
            "url": {
                "raw": "localhost:3000/api/product/254",
                "host": [
                    "localhost"
                ],
                "port": "3000",
                "path": [
                    "api",
                    "product",
                    "254"
                ]
            },
            "description": "Prueba de actualización de productos mediante API REST (simulado)"
        },
        "response": []
    },
    {
        "name": "Eliminamos un producto",
        "request": {
            "method": "DELETE",
            "header": [],
            "body": {
                "mode": "raw",
                "raw": ""
            },
            "url": {
                "raw": "localhost:3000/api/product/576",
                "host": [
                    "localhost"
                ],
                "port": "3000",
                "path": [
                    "api",
                    "product",
                    "576"
                ]
            },
            "description": "Prueba de eliminación de productos mediante el API REST (simulado)"
        },
```

```
            "response": []
        }
    ]
}
```

Nota: nuevamente, antes de continuar, subimos una copia al repositorio:

```
$ git add .
$ git commit -m "API REST CRUD (sin base de datos)"
$ git push
$ git status
```