

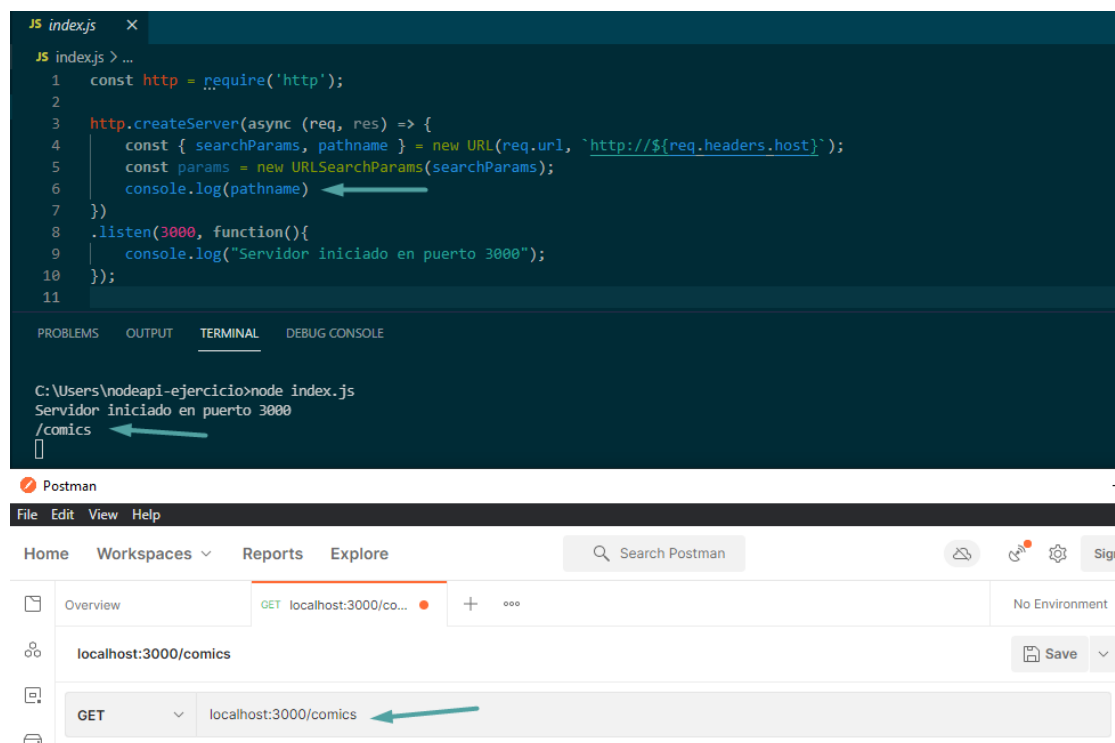
EXERCISES QUE TRABAJAREMOS EN EL CUE:

- EXERCISE 1: PERSISTENCIA DE DATOS EN ARCHIVOS DE TEXTO PLANO.
- EXERCISE 2: PERSISTENCIA DE DATOS EN ARCHIVOS DE TEXTO PLANO II.

EXERCISE 1: PERSISTENCIA DE DATOS EN ARCHIVOS DE TEXTO PLANO.

Ahora que ya hemos repasado los conceptos básicos para crear una API en **node**, los tomaremos aplicaremos utilizando la persistencia de datos en archivos de texto plano.

Crearemos una nueva carpeta que contenga un archivo **index.js**, y dentro de éste escribiremos el mismo código utilizado anteriormente para levantar nuestro servidor; definiremos las variables **URL** y **params**, utilizando **Destructuring**; obtendremos la propiedad **searchParams** para la creación del objeto **URLSearchParams**; y también obtendremos la propiedad **pathname**, la cual contiene la ruta del dominio.



The image shows a development environment with VS Code and Postman. In VS Code, the `index.js` file contains the following code:

```
1  const http = require('http');
2
3  http.createServer(async (req, res) => {
4    const { searchParams, pathname } = new URL(req.url, `http://${req.headers.host}`);
5    const params = new URLSearchParams(searchParams);
6    console.log(pathname)
7  })
8  .listen(3000, function(){
9    console.log("Servidor iniciado en puerto 3000");
10 });
11
```

The terminal output shows the server starting on port 3000 and receiving a request to `/comics`:

```
C:\Users\nodeapi-ejercicio>node index.js
Servidor iniciado en puerto 3000
/comics
```

Below the terminal, the Postman interface is shown. A new request is created with the method **GET** and the URL `localhost:3000/comics`.

Ocuparemos esta propiedad para evaluar si la ruta es correcta, y en conjunto con el método, decidir qué acciones realizará nuestro servidor. En este caso, definiremos, de una vez, los cuatro bloques de código que necesitaremos: GET, POST, PUT, DELETE.

```
JS index.js x
JS index.js > ...
1  const http = require('http');
2
3  http.createServer(async (req, res) => {
4    const { searchParams, pathname } = new URL(req.url, `http://${req.headers.host}`);
5    const params = new URLSearchParams(searchParams);
6
7    if(pathname == '/comics' && req.method == 'GET'){
8
9    }
10
11    if(pathname == '/comics' && req.method == 'POST'){
12
13    }
14
15    if(pathname == '/comics' && req.method == 'PUT'){
16
17    }
18
19    if(pathname == '/comics' && req.method == 'DELETE'){
20
21    }
22  })
23  .listen(3000, function(){
24    console.log("Servidor iniciado en puerto 3000");
25  });
26
27
```

Para la ruta **get**, solo necesitamos leer el archivo, y devolver la información como respuesta. Crearemos un archivo llamado "comics.txt", con el siguiente contenido.

```
1  {
2    "320a1aed-e753-4a94-8eeb-3140236a74e0": {
3      "titulo": "Hombre Araña",
4      "autor": "Stan Lee",
5      "issn": "12341234",
6      "cantidad": 340
7    },
8    "1a266aea-8ef1-48e9-be5a-cfa644f53769": {
9      "titulo": "Capitan America",
10     "autor": "Stan Lee",
11     "issn": "43214321",
12     "cantidad": 230
13   }
14 }
```

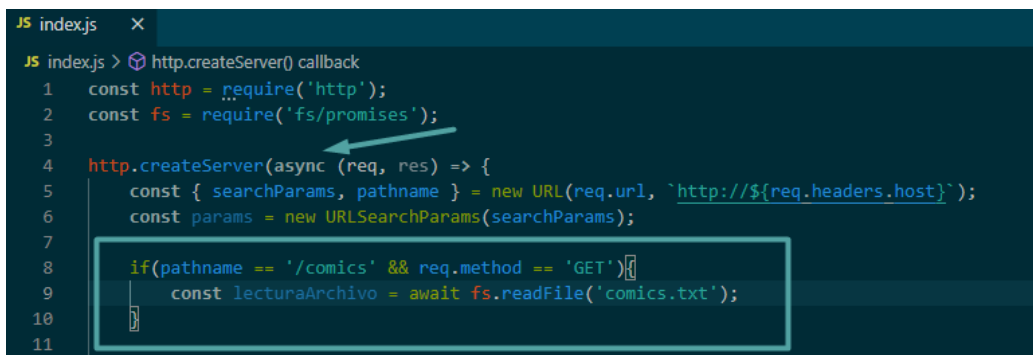
El **string** largo, con números y letras (320a1aed-e753-4a94-8eeb-3140236a74e0), representa el id de acceso para el objeto, el cual utilizaremos para los métodos de actualización y eliminación. Más adelante, emplearemos el paquete **“uuid”** para generarlos automáticamente.

```
comics.txt X
comics.txt
1  {
2    "320a1aed-e753-4a94-8eeb-3140236a74e0":{
3      "titulo": "Hombre Araña",
4      "autor": "Stan Lee",
5      "issn": "12341234",
6      "cantidad": 340
7    },
8    "1a266aea-8ef1-48e9-be5a-cfa644f53769":{
9      "titulo": "Capitan America",
10     "autor": "Stan Lee",
11     "issn": "43214321",
12     "cantidad": 230
13   }
14 }
15
16
```

Ahora, accederemos a los datos de este archivo, utilizando el módulo **fs** dentro de la ruta para el método **GET**. Primero importaremos el módulo; recuerda utilizar la importación para el uso de promesas.

```
JS index.js X
JS index.js > ...
1  const http = require('http');
2  const fs = require('fs/promises');
3
4  http.createServer(async (req, res) => {
5    const { searchParams, pathname } = new URL(req.url, `http://${req.headers.host}`);
6    const params = new URLSearchParams(searchParams);
7
8    if(pathname == '/comics' && req.method == 'GET'){
9
10   }
11
12   if(pathname == '/comics' && req.method == 'POST'){
13
14   }
15
16   if(pathname == '/comics' && req.method == 'PUT'){
17
18   }
19 }
```

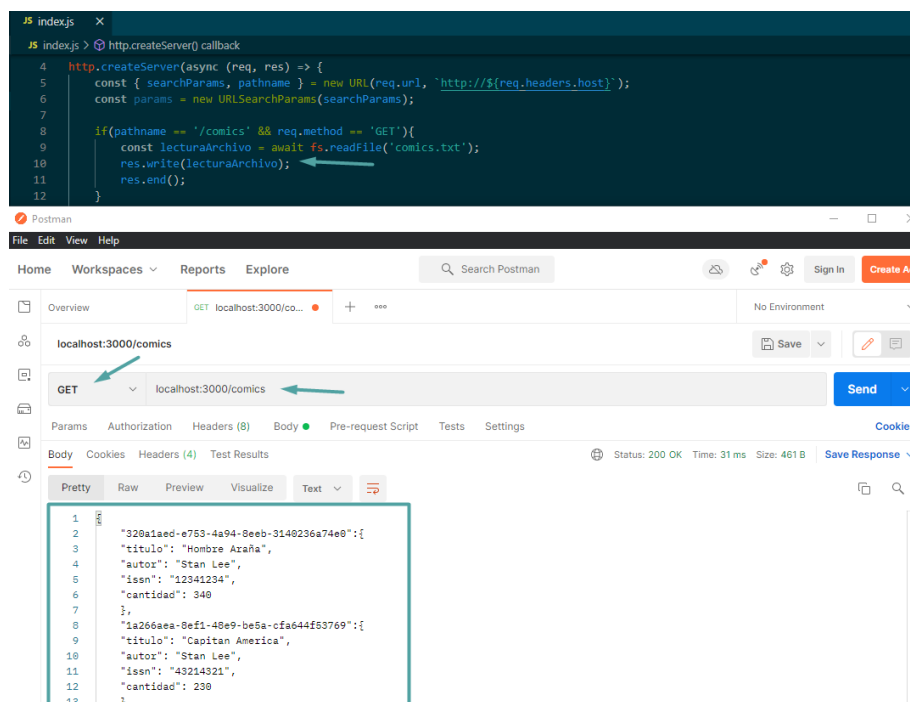
Ahora que ya tenemos importado el módulo, realizamos la lectura de nuestro archivo. Es importante notar que, en este punto, la función del método `createServer()` está siendo utilizada con la palabra clave `async`, lo cual nos permite emplear la palabra clave `await` dentro del cuerpo de esta función.



```

JS index.js x
JS index.js > http.createServer() callback
1  const http = require('http');
2  const fs = require('fs/promises');
3
4  http.createServer(async (req, res) => {
5    const { searchParams, pathname } = new URL(req.url, `http://${req.headers.host}`);
6    const params = new URLSearchParams(searchParams);
7
8    if(pathname == '/comics' && req.method == 'GET'){
9      const lecturaArchivo = await fs.readFile('comics.txt');
10    }
11  })
  
```

Para utilizar el método `res.write()`, y enviar la respuesta a la petición, podemos pasar directamente el resultado de esta lectura sin hacer ninguna transformación de por medio, debido a que el método `write()` es capaz de recibir la cadena de bytes que entrega el método `readFile()` por defecto.



```

JS index.js x
JS index.js > http.createServer() callback
4  http.createServer(async (req, res) => {
5    const { searchParams, pathname } = new URL(req.url, `http://${req.headers.host}`);
6    const params = new URLSearchParams(searchParams);
7
8    if(pathname == '/comics' && req.method == 'GET'){
9      const lecturaArchivo = await fs.readFile('comics.txt');
10     res.write(lecturaArchivo);
11     res.end();
12   }
13 }
  
```

Postman

GET localhost:3000/comics

Status: 200 OK Time: 31 ms Size: 461 B

```

1  {
2    "320a1aed-e753-4a94-8eeb-3149236a74e9": {
3      "titulo": "Hombre Araña",
4      "autor": "Stan Lee",
5      "issn": "12341234",
6      "cantidad": 340
7    },
8    "1a266aea-8ef1-48e9-be5a-cfa644f53769": {
9      "titulo": "Capitan America",
10     "autor": "Stan Lee",
11     "issn": "43214321",
12     "cantidad": 230
13   }
  
```

Para la petición de tipo post, primero debemos leer el archivo, pero esta vez lo transformaremos a objeto, utilizando `JSON.parse()`.

```

13
14     if(pathname == '/comics' && req.method == 'POST'){
15         const archivoOriginal = await fs.readFile('comics.txt');
16         const datosOriginales = JSON.parse(archivoOriginal);
17         console.log(datosOriginales);
18     }
19

```

PROBLEMS OUTPUT **TERMINAL** DEBUG CONSOLE

```

C:\Users\nodeapi-ejercicio>node index.js
Servidor iniciado en puerto 3000
{
  '320a1aed-e753-4a94-8eeb-3140236a74e0': {
    titulo: 'Hombre Araña',
    autor: 'Stan Lee',
    issn: '12341234',
    cantidad: 340
  },
  '1a266aea-8ef1-48e9-be5a-cfa644f53769': {
    titulo: 'Capitan America',
    autor: 'Stan Lee',
    issn: '43214321',
    cantidad: 230
  }
}

```

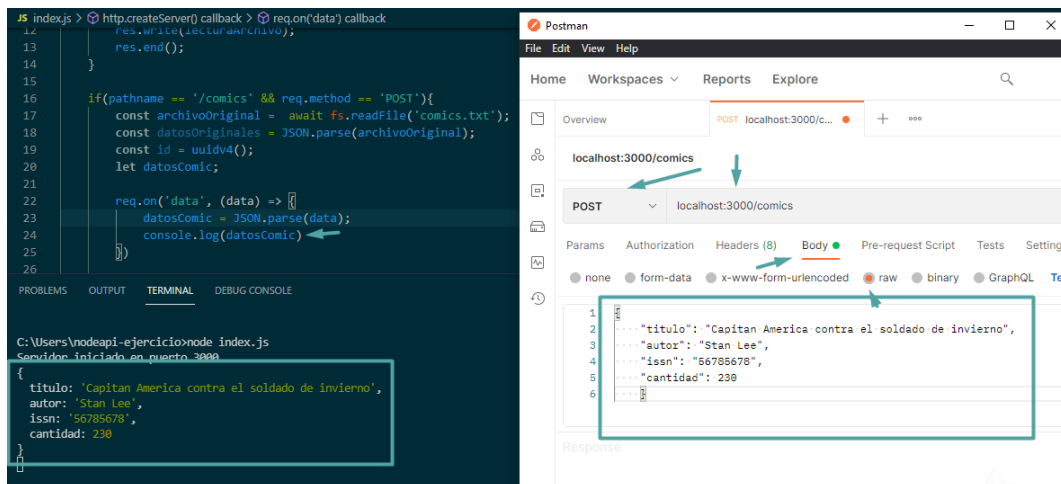
Ahora que ya tenemos nuestra información original, primero crearemos un id único para el objeto nuevo. Utilizaremos el comando `"npm install uuid"`, y luego requeriremos el paquete de la siguiente forma, tal como lo indica la documentación (<https://www.npmjs.com/package/uuid>).

```

JS index.js  X  comics.txt
JS index.js > ...
1  const http = require('http');
2  const fs = require('fs/promises');
3  const { v4: uuidv4 } = require('uuid');
4
5

```

Recibiremos los datos desde la petición utilizando el método `req.on('data')`, y los transformaremos a un objeto con `JSON.parse()`.



The screenshot shows two windows. On the left, VS Code displays the `index.js` file with the following code:

```

12 http.createServer() callback > req.on('data') callback
13   res.writeHead(200, { 'Content-Type': 'text/plain' });
14   res.end();
15 }
16 if (pathname === '/comics' && req.method === 'POST') {
17   const archivoOriginal = await fs.readFile('comics.txt');
18   const datosOriginales = JSON.parse(archivoOriginal);
19   const id = uuidv4();
20   let datosComic;
21
22   req.on('data', (data) => {
23     datosComic = JSON.parse(data);
24     console.log(datosComic);
25   });
26 }

```

The terminal output shows the server starting on port 3000 and receiving a POST request with the following data:

```

{
  titulo: 'Capitan America contra el soldado de invierno',
  autor: 'Stan Lee',
  issn: '56785678',
  cantidad: 230
}

```

On the right, Postman shows a POST request to `localhost:3000/comics` with a raw body containing the same JSON data:

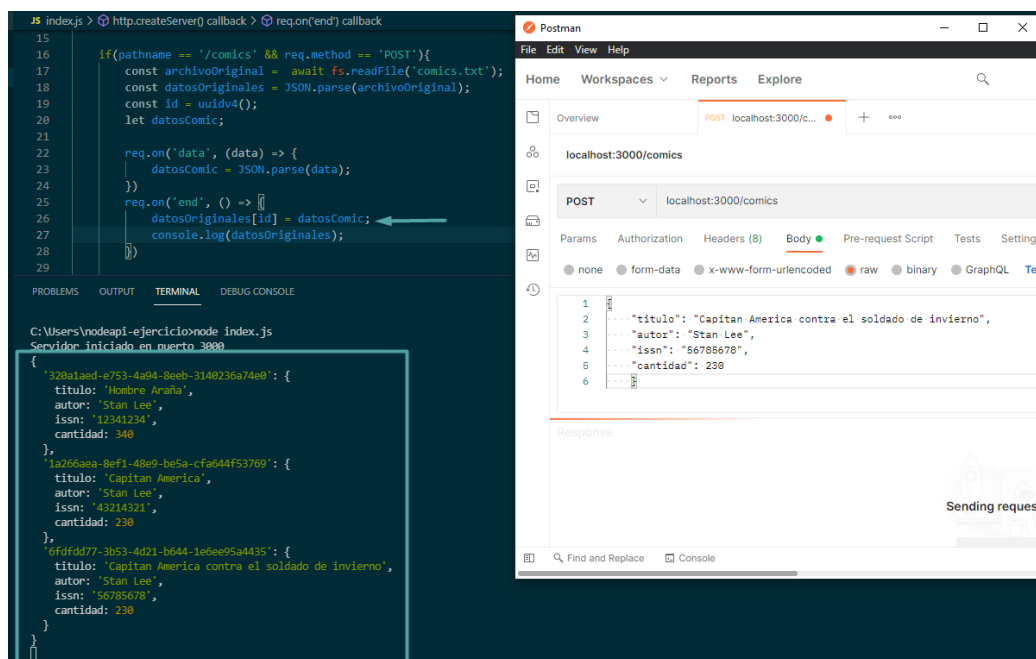
```

{
  "titulo": "Capitan America contra el soldado de invierno",
  "autor": "Stan Lee",
  "issn": "56785678",
  "cantidad": 230
}

```

Ya que tenemos la información en un objeto, solo nos queda agregar el nuevo, y guardar esto en el archivo.

En primer lugar, agregaremos la nueva propiedad utilizando el id generado y la información recibida, con lo que ya tenemos el nuevo comic ingresado en el objeto.



The screenshot shows two windows. On the left, VS Code displays the `index.js` file with the following code:

```

15 http.createServer() callback > req.on('end') callback
16   res.writeHead(200, { 'Content-Type': 'text/plain' });
17   res.end();
18 if (pathname === '/comics' && req.method === 'POST') {
19   const archivoOriginal = await fs.readFile('comics.txt');
20   const datosOriginales = JSON.parse(archivoOriginal);
21   const id = uuidv4();
22   let datosComic;
23
24   req.on('data', (data) => {
25     datosComic = JSON.parse(data);
26   });
27   req.on('end', () => {
28     datosOriginales[id] = datosComic;
29     console.log(datosOriginales);
30   });
31 }

```

The terminal output shows the server starting on port 3000 and receiving a POST request with the following data:

```

{
  '328a1aed-e753-4a94-8eeb-3140236a74e0': {
    titulo: 'Homre Araña',
    autor: 'Stan Lee',
    issn: '12341234',
    cantidad: 340
  },
  '1a266aea-8ef1-48e9-be5a-cfa644f53769': {
    titulo: 'Capitan America',
    autor: 'Stan Lee',
    issn: '43214321',
    cantidad: 230
  },
  '0fd4f477-3b53-4d21-b644-1e6ee95a4435': {
    titulo: 'Capitan America contra el soldado de invierno',
    autor: 'Stan Lee',
    issn: '56785678',
    cantidad: 230
  }
}

```

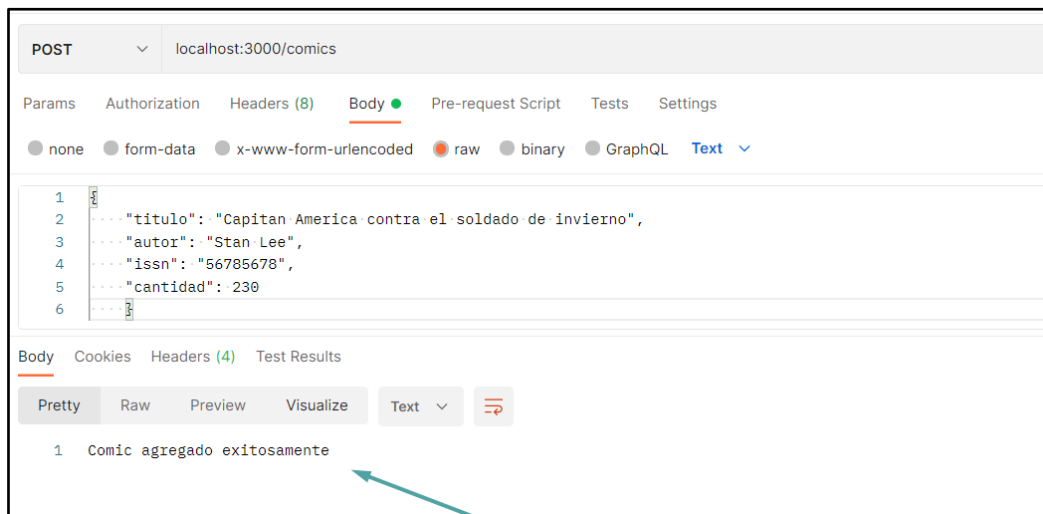
On the right, Postman shows a POST request to `localhost:3000/comics` with a raw body containing the same JSON data as before.

Y ahora solo nos queda utilizar este objeto para escribir la información en el archivo comic.txt.

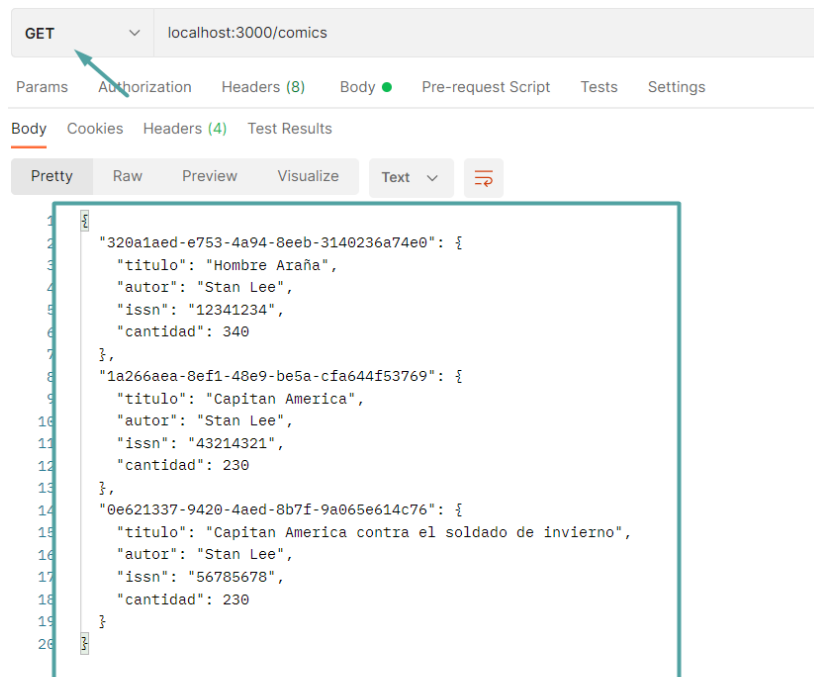
```
if(pathname == '/comics' && req.method == 'POST'){
  const archivoOriginal = await fs.readFile('comics.txt');
  const datosOriginales = JSON.parse(archivoOriginal);
  const id = uuidv4();
  let datosComic;

  req.on('data', (data) => {
    datosComic = JSON.parse(data);
  })
  req.on('end', async () => {
    datosOriginales[id] = datosComic;
    await fs.writeFile('comics.txt', JSON.stringify(datosOriginales, null, 2));
    res.write("Comic agregado exitosamente");
    res.end()
  })
}
```

Al hacer la consulta mediante **Postman**, obtenemos.



Y si realizamos una consulta de tipo **get**, veremos el nuevo comic añadido. También puedes revisar tu archivo comics.txt de la siguiente forma: `localhost:3000/comics?id=320a1aed-e753-4`



EXERCISE 2: PERSISTENCIA DE DATOS EN ARCHIVOS DE TEXTO PLANO II.

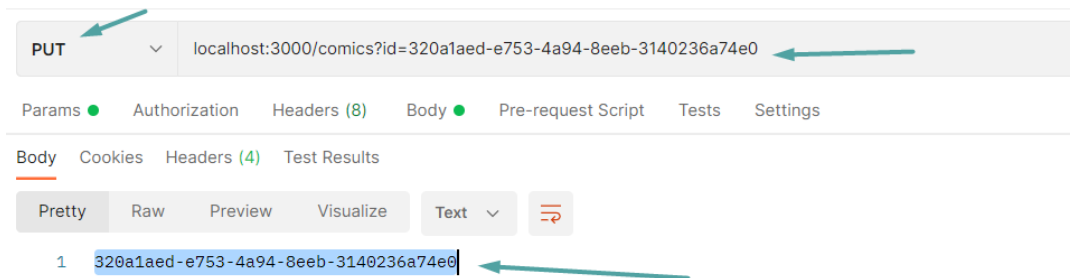
Para actualizar un comic, enviaremos su id por medio de **query strings**. Solo debes ir al archivo `comics.txt`, y copiar el id de un comic; utilizaremos el del primer comic para el siguiente bloque de código. Por lo tanto, la URL a consultar quedaría de la siguiente forma:

localhost:3000/comics?id=320a1aed-e753-4a94-8eeb-3140236a74e0

Para obtener la URL dentro de nuestro programa, solo necesitamos acceder al id utilizando el método `get('id')` de la variable `params` definida anteriormente

```
if(pathname == '/comics' && req.method == 'PUT'){
  res.write(params.get('id'));
  res.end();
}
```

Y si vemos la respuesta en **postman**, podemos observar que tenemos de vuelta nuestro **id**.



Ahora que ya sabemos cómo obtener el id en nuestro servidor, primero leeremos el archivo `comics.txt`, y guardaremos el resultado en un objeto; además, agregaremos el método `on()` para recibir los datos, guardarlos en una variable, y luego trabajarlos una vez haya terminado la petición.

```
if(pathname == '/comics' && req.method == 'PUT'){
  const datosArchivo = await fs.readFile('comics.txt');
  const objetoArchivoOriginal = JSON.parse(datosArchivo);
  let datosParaModificar;
  req.on('data', (datos) => {
    datosParaModificar = JSON.parse(datos);
  })
  req.on('end', () => {
    //Codigo para actualizar los datos
  })
}
```

Para modificar el archivo, primero accederemos al objeto que se actualizará, utilizando el id obtenido, y luego, utilizando el `spread operator`, le agregaremos los campos nuevos que han llegado en el body de la petición.

```
if(pathname == '/comics' && req.method == 'PUT'){
  const id = params.get('id');
  const datosArchivo = await fs.readFile('comics.txt');
  const objetoArchivoOriginal = JSON.parse(datosArchivo);
  let datosParaModificar;
  req.on('data', (datos) => {
    datosParaModificar = JSON.parse(datos);
  })
  req.on('end', () => {
    const comicOriginal = objetoArchivoOriginal[id]
    const comicActualizado = {...comicOriginal, ...datosParaModificar }
    res.write(JSON.stringify(comicActualizado, null, 2));
    res.end();
  })
}
```

Si vemos la respuesta de este código, obtenemos el objeto actualizado.

The screenshot shows a REST client interface with a PUT request to `localhost:3000/comics?id=320a1aed-e753-4a94-8eeb-3140236a74e0`. The request body is a JSON object: `{ "titulo": "El intrepido hombre araña" }`. The response body is a JSON object: `{ "titulo": "El intrepido hombre araña", "autor": "Stan Lee", "issn": "12341234", "cantidad": 340 }`.

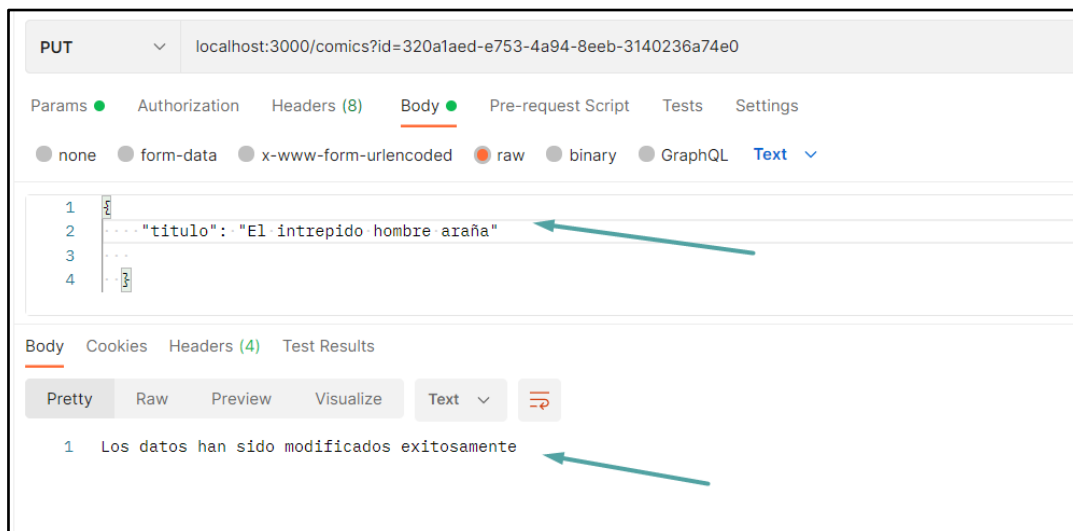
Ahora solo nos queda modificar el objeto original con las nuevas propiedades, y luego escribir los datos a nuestro archivo `comics.txt`.

```
if(pathname == '/comics' && req.method == 'PUT'){
  const id = params.get('id');
  const datosArchivo = await fs.readFile('comics.txt');
  const objetoArchivoOriginal = JSON.parse(datosArchivo);
  let datosParaModificar;
  req.on('data', (datos) => {
    datosParaModificar = JSON.parse(datos);
  })
  req.on('end', async () => {
    const comicOriginal = objetoArchivoOriginal[id]
    const comicActualizado = {...comicOriginal, ...datosParaModificar }

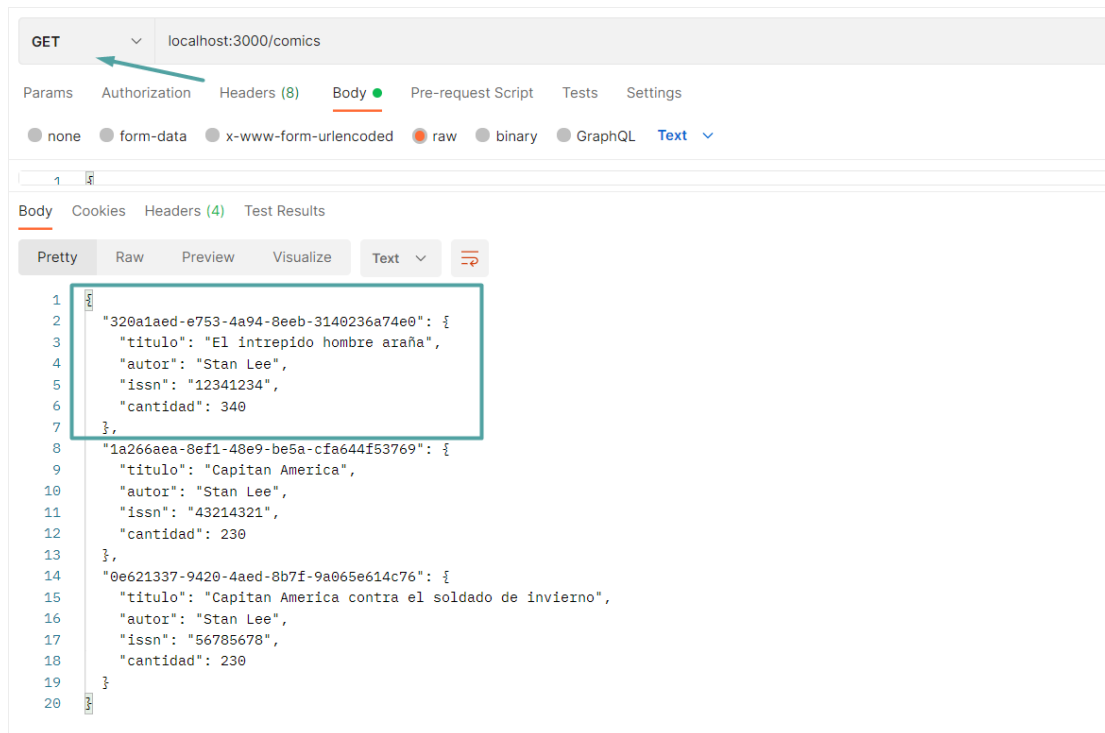
    objetoArchivoOriginal[id] = comicActualizado;
    await fs.writeFile('comics.txt', JSON.stringify(objetoArchivoOriginal, null, 2));

    res.write("Los datos han sido modificados exitosamente");
    res.end();
  })
}
```

Realizando la petición desde **Postman**, podemos observar la respuesta que confirma la modificación de los datos.



Y si realizamos una petición de tipo **get** para obtener todos los datos, podemos observar que el archivo ha sido modificado. También puedes revisar el archivo de texto comics.txt.

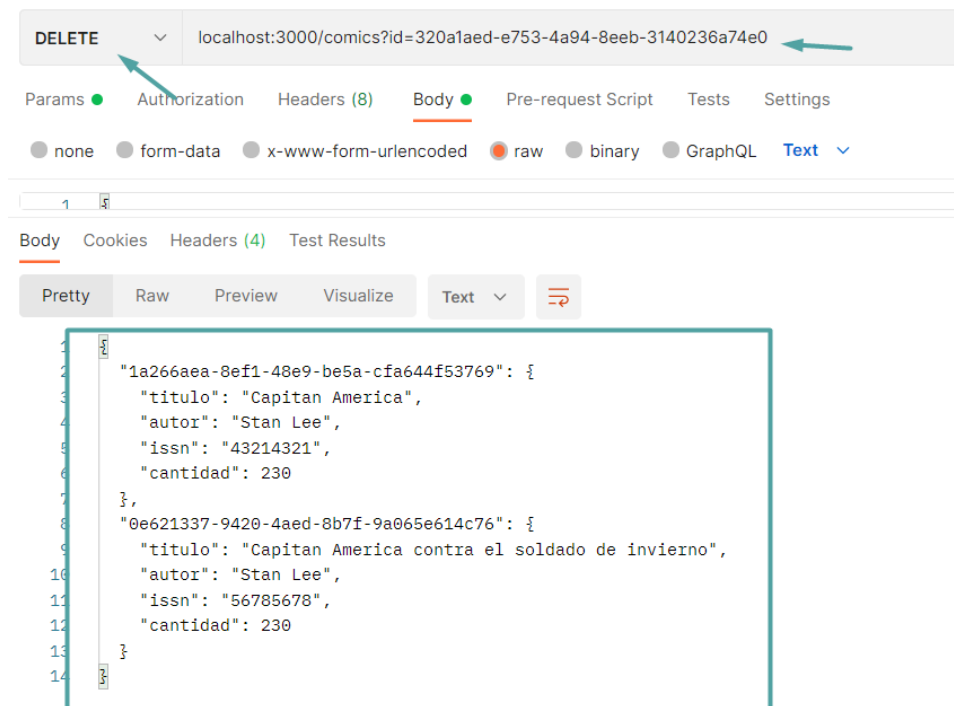


Por último, para la ruta de tipo **DELETE**, también utilizaremos el id que será enviado por la **URL** mediante **query strings**. Al igual que en los casos anteriores, primero obtendremos la información que contiene el archivo original, y la guardaremos como objeto dentro de una variable; además, definiremos la variable **id** que contendrá el id obtenido de la URL, utilizando el método **get** en el objeto **params**. Luego, lo utilizaremos para eliminar el comic del objeto **"objetoComicsOriginal"**.

```

if(pathname == '/comics' && req.method == 'DELETE'){
  const comicsOriginales = await fs.readFile('comics.txt');
  const objetoComicsOriginal = JSON.parse(comicsOriginales);
  const id = params.get('id');
  delete objetoComicsOriginal[id];
  res.write(JSON.stringify(objetoComicsOriginal, null, 2));
  res.end();
}
  
```

Utilizando el mismo id del primer comic para la petición desde **postman**, recibimos como respuesta el objeto modificado, el cual no contiene el primer comic.



Ya solo nos queda utilizar este objeto para escribir el contenido del archivo de texto.

```

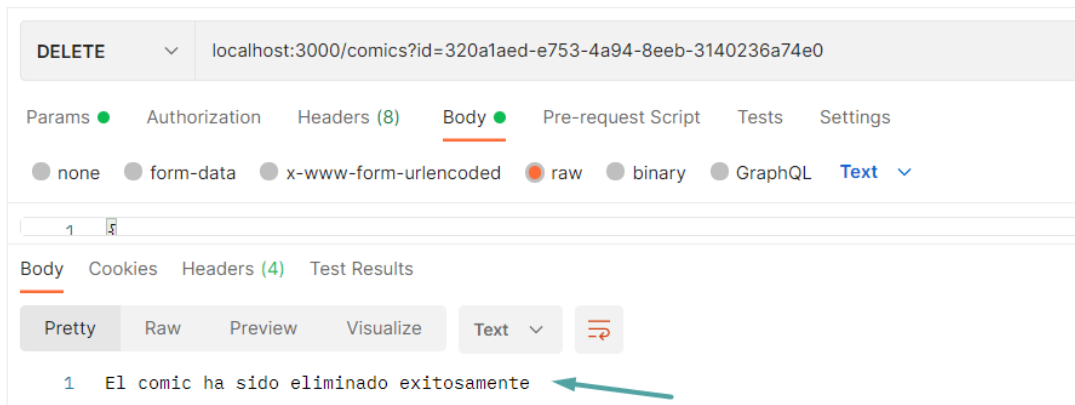
if(pathname == '/comics' && req.method == 'DELETE'){
  const comicsOriginales = await fs.readFile('comics.txt');
  const objetoComicsOriginal = JSON.parse(comicsOriginales);
  const id = params.get('id');
  delete objetoComicsOriginal[id];

  await fs.writeFile('comics.txt', JSON.stringify(objetoComicsOriginal, null, 2));

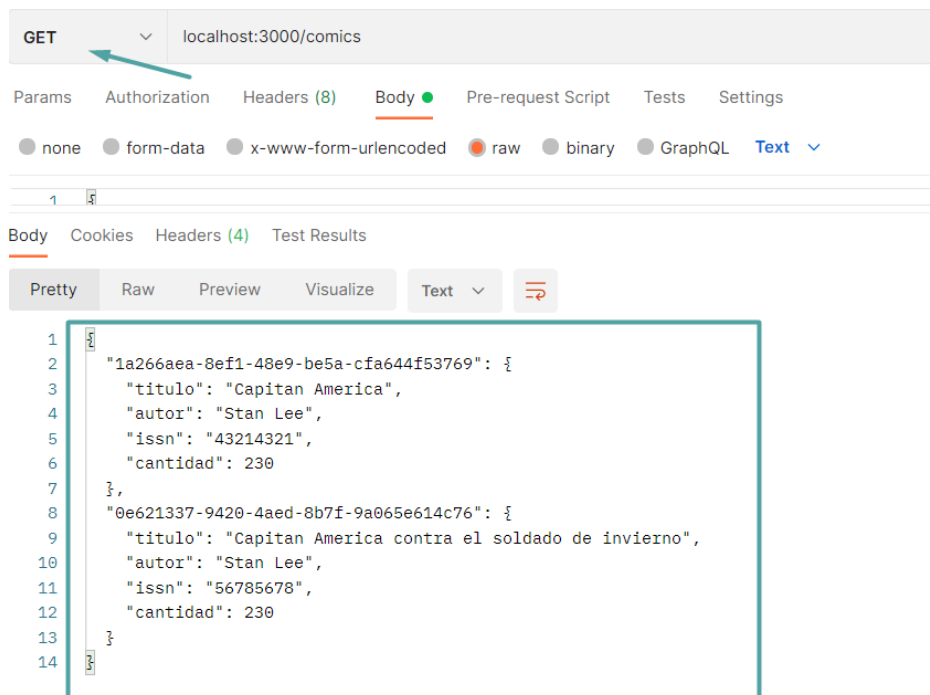
  res.write("El comic ha sido eliminado exitosamente");
  res.end();
}

```

Y luego, al realizar la petición con **postman**, obtendremos la confirmación respecto al comic eliminado.



Para confirmar esto, podemos consultar nuevamente la ruta con el método **get**, y veremos como el primer comic ha sido eliminado; también puede consultar el archivo de texto comics.txt.



Ya en este punto has logrado construir una **API Restful**, con funciones **CRUD** y persistencia de datos desde cero. Si bien existen partes del código que son extensas, todo esto se reducirá cuando profundices más en el framework Express, que como lo hemos visto anteriormente, se encarga de lidiar con varias tareas de forma automática, las cuales hasta ahora habíamos desarrollado manualmente.