

EXERCISES QUE TRABAJAREMOS EN LA CUE

- EXERCISE 1: SUBIENDO ARCHIVOS USANDO EXPRESS

EXERCISE 1: SUBIENDO ARCHIVOS USANDO EXPRESS

El objetivo del presente ejercicio es una guía paso a paso de continuación con el CUE donde se procederá a la consulta de listados de archivos por medio del método **GET**, descarga y eliminación de un archivo en el servidor

Pasos principales que se realizarán para la carga de archivos en el servidor (To-Do):

- 1- Listado de los archivos cargados en el servidor, método **GET /files**
- 2- Descarga de un archivo del servidor, método **GET /files:name**
- 3- Eliminación de archivos en el Servidor, método **DELETE /files/:name**

1. LISTADO DE LOS ARCHIVOS CARGADOS EN EL SERVIDOR, MÉTODO GET /FILES

La visualización o listado de los archivos subidos al servidor, procedemos a crear un método para el mismo, para ello utilizamos el método **GET** que me permitirá traer el el conjunto de archivos. Utilizaremos el método **fs.readdir()** se utiliza para leer de forma asíncrona el contenido de un directorio determinado. La devolución de llamada de este método devuelve una matriz de todos los nombres de archivo en el directorio. El siguiente código refleja el listado de los archivos subidos al servidor:

```
1 app.get('/files', async (req, res) => {
2   const directoryPath = "./files/";
3
4   // El método fs.readdir() se utiliza para leer de forma
5   // asíncrona el contenido de un directorio determinado.
6   fs.readdir(directoryPath, function(err, files) {
7     if (err) {
8       res.status(500).send({
9         message: "No se puede buscar archivos en el directorio!",
10      });
11    }
12    // Variable que contiene el listado de archivos en el servidor
13
14    let listFiles = [];
```

```
15     files.forEach((file) => {  
16         listFiles.push({  
17             name: file,  
18             url: baseUrl + file,  
19         });  
20     });  
21     res.status(200).send(listFiles);  
22 });  
23 });
```

A partir de la línea 8 podemos ver que nuestro objeto de respuesta contiene tanto un código HTTP como un mensaje que podemos enviar como retroalimentación al usuario. En este caso, estos parámetros están configurados para que Express devuelva un código de estado HTTP 500, "Error interno del servidor", con un mensaje de estado que dice "No se puede buscar archivos en el directorio". Si deseamos retornar otros mensajes de estado al cliente debemos hacerlo mediante el atributo `message` del método `send`.

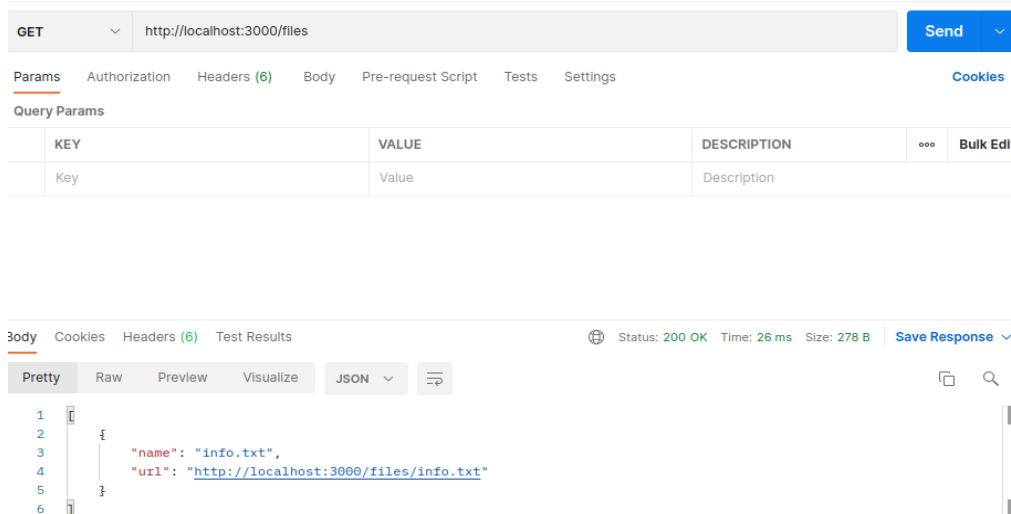
Al ejecutar en el navegador o en Postman observamos que no tenemos respuesta y en la consola del terminal verificamos que `fs` no está definida.

```
[nodemon] restarting due to changes...  
[nodemon] starting 'node index.js'  
Corriendo en el servidor, API REST subida de archivos express-fileupload que se está ejecutando en: http://localhost:3000.  
(node:17565) UnhandledPromiseRejectionWarning: ReferenceError: fs is not defined
```

Para ello debemos agregar al inicio de nuestro script o archivo `index.js` las librerías requeridas y agregamos también la variable `baseUrl` que tampoco está definida:

```
1 const express = require('express')  
2 const fileUpload = require('express-fileupload')  
3 const util = require('util');  
4 const fs = require('fs');  
5 const baseUrl = "http://localhost:3000/files/";
```

Al ejecutar nuevamente Postman o en navegador observamos la lista de archivos en el servidor:



GET http://localhost:3000/files Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

body Cookies Headers (6) Test Results Status: 200 OK Time: 26 ms Size: 278 B Save Response

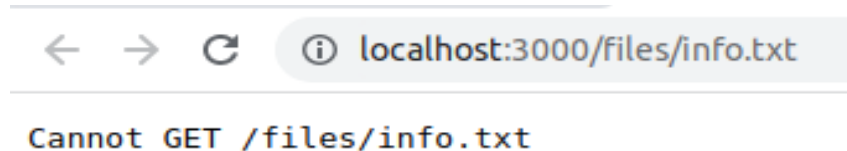
Pretty Raw Preview Visualize JSON

```

1  {
2    "name": "info.txt",
3    "url": "http://localhost:3000/files/info.txt"
4  }
5
6
  
```

2. DESCARGA DE UN ARCHIVO DEL SERVIDOR, MÉTODO GET /FILES:NAME

El método anterior nos permite listar el conjunto de archivos, mas no podemos descargarlos u obtenerlos por medio del método **GET** ya que no lo hemos definido.



Procedemos a crear el método **GET** para descargar un archivo:

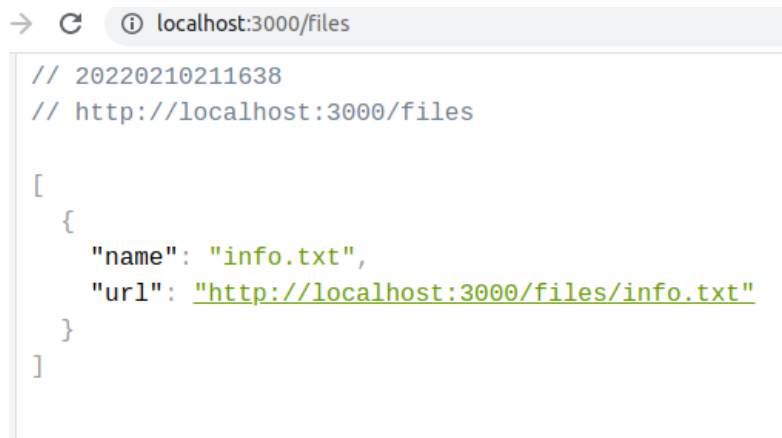
```

1 app.get('/files/:name', async (req, res) => {
2   const fileName = req.params.name;
3   const directoryPath = "./files/";
4   // La función res.download() transfiere el archivo en la ruta
5   // como un "archivo adjunto". Por lo general, los navegadores
6   // le pedirán al usuario que descargue.
7   res.download(directoryPath + fileName, fileName, (err) => {
8     if (err) {
9       res.status(500).send({
10        message: "No se puede descargar el archivo. " + err,
11      });
12    }
  })
}
  
```

```
13     });  
14 });
```

Si nuestro método incurre en un error, podemos enviar un mensaje de error, o de estado, al cliente usando el objeto respuesta `res`, incluyendo detalles como el código de estatus y un mensaje personalizado.

Verificamos en el navegador la descarga del archivo al hacer clic en el enlace:
<http://localhost:3000/files/info.txt>



```
→ ↻ localhost:3000/files  
// 20220210211638  
// http://localhost:3000/files  
  
[  
  {  
    "name": "info.txt",  
    "url": "http://localhost:3000/files/info.txt"  
  }  
]
```

3. ELIMINACIÓN DE ARCHIVOS EN EL SERVIDOR, MÉTODO DELETE /FILES/:NAME

Para la eliminación de un archivo en el servidor utilizaremos la función `fs.unlinkSync` que elimina un archivo y espera hasta que se termine la operación para seguir ejecutando el código.

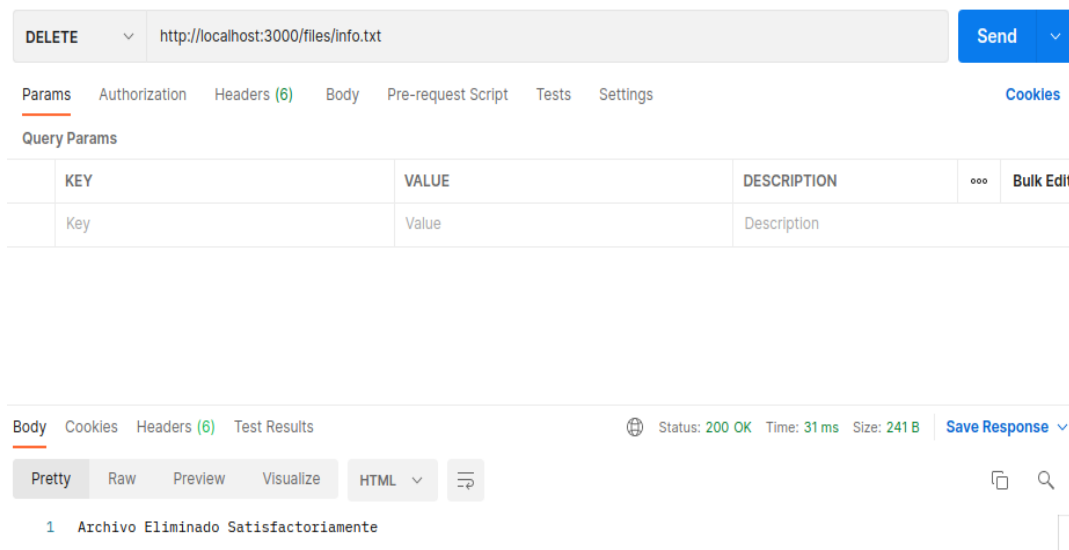
Esta eliminación se realiza por medio del método **DELETE** a través de la ruta `/files/:name`, siendo `name` el nombre del archivo a eliminar. La estructura de método es la siguiente:

```
1 app.delete('/files/:name', async (req, res) => {  
2   const fileName = req.params.name;  
3   const directoryPath = './files/';  
4   try {  
5     // fs.unlinkSync elimina un archivo y espera hasta que se termine  
6     la  
7     // operación para seguir ejecutando el código, también se puede  
8     // usar fs.unlink() que ejecuta dicha operación de forma  
9     asincrónica
```

```

10     fs.unlinkSync(directoryPath + fileName);
11     console.log('File removed')
12     res.status(200).send("Archivo Eliminado Satisfactoriamente");
13   } catch (err) {
14     console.error('ocurrió algo incorrecto al eliminar el archivo',
15 err)
16   }
17 });
  
```

Verificamos en Postman



DELETE ▼ http://localhost:3000/files/info.txt Send ▼

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (6) Test Results ⊕ Status: 200 OK Time: 31 ms Size: 241 B Save Response ▼

Pretty Raw Preview Visualize HTML ▼ 🔍

1 Archivo Eliminado Satisfactoriamente

Y en la consola de terminal:

```

[nodemon] watching extensions: js,mjs,json
[nodemon] starting 'node index.js'
Corriendo en el servidor, API REST subida de archivos express-fileupload que se esta ejecutando en: http://localhost:3000.
File removed
  
```

Si intentamos nuevamente eliminar el archivo en el servidor, verificamos que el cliente REST Postman se queda resolviendo sin mensaje de respuesta al cliente, porque no la hay al no encontrar el archivo en el servidor y ademas observamos en la consola del terminal que nos refleja el error de archivo no encontrado en el directorio.

```

[nodemon] watching extensions: js,mjs,json
[nodemon] starting 'node index.js'
Corriendo en el servidor, API REST subida de archivos express-fileupload que se esta ejecutando en: http://localhost:3000.
ocurrió algo incorrecto al eliminar el archivo: ENOENT: no such file or directory, unlink './files/info.txt'
  
```

Para validar la existencia del archivo a eliminar, debemos buscar en el directorio por medio de la función `fs.readdir`, si encuentra el archivo en el directorio lo elimina caso contrario emite un mensaje al cliente que no se encontró el archivo a eliminar en el servidor.

```
1 app.delete('/files/:name', async (req, res) => {
2   const fileName = req.params.name;
3   const directoryPath = "./files/";
4   let listFiles = [];
5   try {
6     fs.readdir(directoryPath, function(err, files) {
7       if (err) {
8         res.status(500).send({
9           message: "No se puede buscar archivos en el
10 directorio!",
11         });
12       }
13       // Variable que contiene el listado de archivos en el servidor
14
15       files.forEach((file) => {
16         listFiles.push(file);
17       });
18       // verificamos si el archivo se encuentra en el directorio
19       let fileBusqueda = listFiles.find(l => l === fileName);
20       if (!fileBusqueda) {
21         return res.status(409).json({
22           message: 'No se encontró el archivo a eliminar en el
23 servidor'
24         });
25       } else {
26         // fs.unlinkSync elimina un archivo y espera hasta que se
27 termine la
28         // operación para seguir ejecutando el código, también se
29 puede
30         // usar fs.unlink() que ejecuta dicha operación de forma
31 asíncrona
32         fs.unlinkSync(directoryPath + fileName);
33         console.log('Archivo Eliminado');
34         res.status(200).send("Archivo Eliminado
35 Satisfactoriamente");
36       }
37     });
38   } catch (err) {
39     console.error('ocurrió algo incorrecto al eliminar el archivo',
40 err)
41   }
42 }
```

```
43 } )
```

NOTA:

Para una mejor resolución del proyecto, se recomienda tener actualizado la versión de Node.js a la última versión y codificar con ES6.