

EXERCISES QUE TRABAJAREMOS EN EL CUE

- EXERCISE 1: IMPLEMENTAR LA GESTIÓN DE SUBIDA DE ARCHIVOS A UN SERVIDOR HACIENDO USO DE EXPRESS-FILEUPLOAD.

EXERCISE 1: IMPLEMENTAR LA GESTIÓN DE SUBIDA DE ARCHIVOS A UN SERVIDOR HACIENDO USO DE EXPRESS-FILEUPLOAD

El objetivo de este ejercicio es plantear una guía paso a paso donde se procederá a cargar el archivo al servidor, por medio del método **POST**, consulta de listados de archivos por medio del método GET, descarga y eliminación de un archivo en el servidor.

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

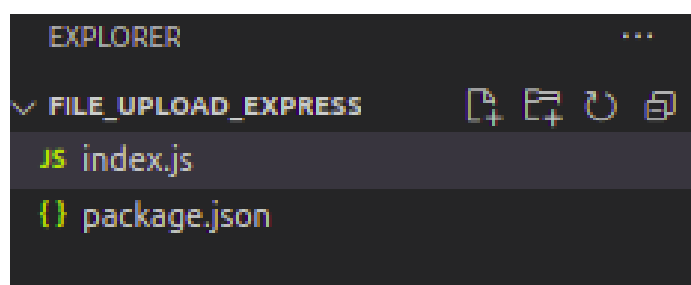
- 1- Crear una nueva aplicación Node Js.
- 2- Instalar Express, Body parse, cors y express-fileupload.
- 3- Crear el archivo index.js.
- 4- Ejecutar la aplicación de carga de archivos en Express de Node js, método POST/upload.
- 5- Validar la subida de archivo no nulo, según el tipo de archivo.

1. CREAR UNA NUEVA APLICACIÓN NODE JS

Para iniciar la configuración del proyecto, debemos buscar la terminal estando ubicados en una carpeta a elección, y seguidamente realizamos los siguientes pasos:

```
1 $ Crear el proyecto
2 $ mkdir file_upload_express
3 $ Accedemos a la carpeta
4 $ mkdir crear_api_node_express
5 $ Inicializar el proyecto con los parámetros por defecto
6 $ npm init -y
```

Procedemos a crear el archivo index.js:



2. INSTALE EXPRESS, BODY PARSE, CORS Y EXPRESS-FILEUPLOAD

Procedemos a instalar Express.js y express-fileupload, que es la dependencia que se utilizará para la gestión y subida de archivos al servidor en back-end para Node.js. Para instalarlo en el proyecto se ejecuta:

```
1 $ Instalando Express.js
2 $ npm install --save express
3
4 $ Instalando express-fileupload
5 $ npm install express-fileupload
6
7 $ Instalamos el nodemon
8 $ npm i nodemon -D
```

Podemos observar el archivo package.json:

```
{ package.json X
  {} package.json > {} devDependencies
  1  {
  2    "name": "uploadfiletoserver",
  3    "version": "1.0.0",
  4    "description": "",
  5    "main": "index.js",
  6    "scripts": {
  7      "test": "echo \"Error: no test specified\" && exit 1",
  8      "start": "nodemon index.js"
  9    },
 10    "keywords": [],
 11    "author": "",
 12    "license": "ISC",
 13    "dependencies": {
 14      "body-parser": "^1.19.1",
 15      "cors": "^2.8.5",
 16      "express": "^4.17.2",
 17      "express-fileupload": "^1.3.1",
 18      "lodash": "^4.17.21",
 19      "morgan": "^1.10.0"
 20    },
 21    "devDependencies": {
 22      "nodemon": "^2.0.15"
 23    }
 24  }
 25 }
```

Agregamos al mismo en la sección de script:

```
1 "start": "nodemon index.js"
```

Para luego, ejecutar el script con:

```
1 $ npm run star
```

3. CREAR EL ARCHIVO INDEX.JS

Una vez configurado el entorno, editaremos el archivo index.js para agregar las librerías correspondientes, y escribiremos el siguiente código:

```
// Cargando el modulo de Express.js
1 const express = require('express')
2
3 // Cargando la librería de express-fileupload
4 const fileUpload = require('express-fileupload')
5
6 // Este variable define el puerto del computador donde la API esta
7 disponible
8 const PORT = 3000;
9
10 // Definimos la variable que inicializa la libreria Express.js
11 const app = express();
12
13 // Middleware
14 app.use(fileUpload({
15   createParentPath: true
16 }));
17
18 // 1 - El puerto donde esta disponible la API
19 // 2 - Una función de llamada (callback) cuando la API esta lista
20 app.listen(PORT, () =>
21   console.log(`Corriendo en el servidor, API REST subida de archivos
22   express-fileupload que se esta ejecutando en: http: //localhost:${PORT}.`)
23 );
```

Se ejecuta en la consola o terminal el siguiente comando:

```
1 $ npm run start
```

```
[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[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.
```

4. EJECUTE LA APLICACIÓN DE CARGA DE ARCHIVOS EN EXPRESS DE NODE JS, MÉTODO POST /UPLOAD

Procedemos a crear el método de carga con el siguiente código:

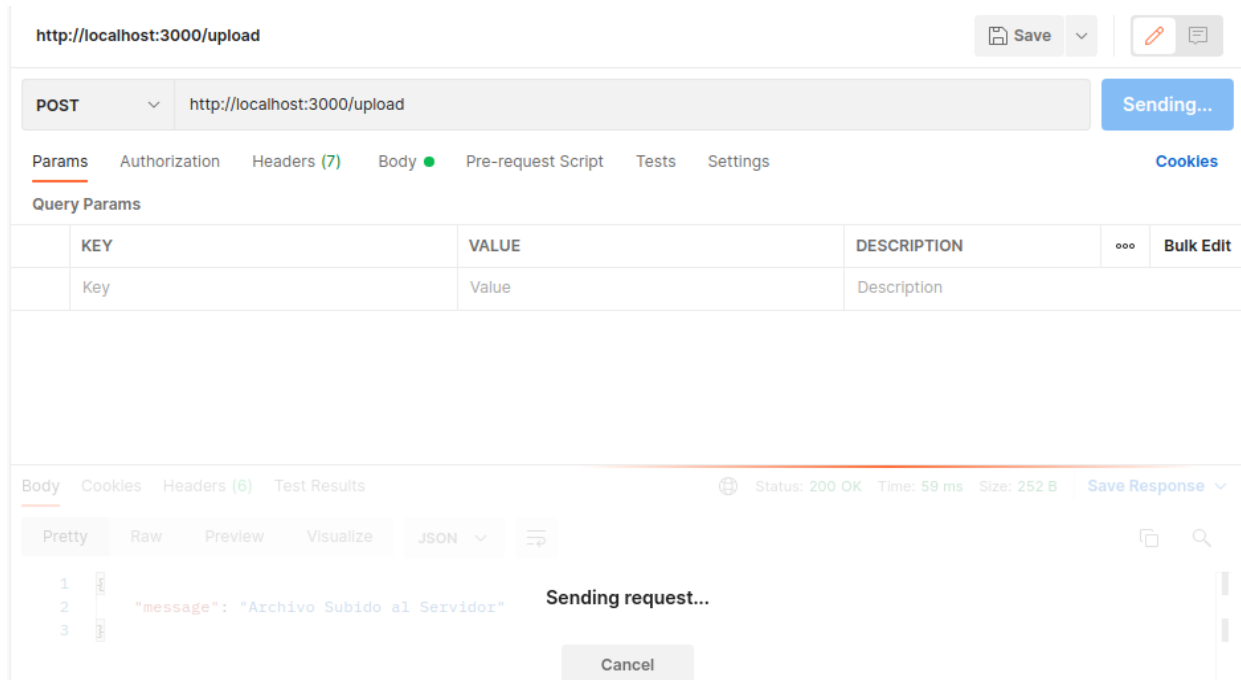
```
1 app.post('/upload', async (req, res) => {
2     let fileRecived = req.files.fileName;
3     let extName = fileRecived.name;
4     uploadPath = './files/' + extName;
5
6     fileRecived.mv(uploadPath, err => {
7         if (err) {
8             return res.status(500).send({
9                 message: err
10            });
11        }
12        return res.status(200).send({
13            message: 'Archivo Subido al Servidor'
14        });
15    });
```

fileRecived: se asigna la solicitud **req.files.fileName**, siendo el **fileName** la variable que posee el el archivo al enviar la solicitud.

extName: variable que posee el nombre real del archivo, que se obtiene de la estructura **name**.

uploadPath: es la ruta en el servidor donde se desea guardar el archivo, en este caso, en el directorio actual en la carpeta files con el mismo nombre que estamos subiendo al servidor.

Ejecutamos las pruebas con Postman, utilizando el método **POST**, y en la ruta: `http://localhost:3000/upload`



Observamos que nos presenta un error en la consola del terminal:

```
[nodemon] restarting due to changes...  
[nodemon] starting 'node index.js'  
Corriendo en el servidor, API REST subida de archivos express-fileupload que se esta ejecutando en: http://localhost:3000.  
(node:28974) UnhandledPromiseRejectionWarning: TypeError: Cannot read property 'fileName' of undefined
```

No hemos definido la propiedad o la variable **fileName** del tipo archivo, esto es en Postman, Body → form-data, en el campo **KEY** colocamos **fileName**, y en **VALUE** seleccionamos un archivo y enviamos la solicitud **POST**.

5. VALIDANDO LA SUBIDA DE ARCHIVO NO NULO, SEGÚN EL TIPO DE ARCHIVO

La validación de archivo no nulo se realiza por medio de una condicional, la cual verifica la existencia del mismo por medio de la solicitud, o verificando que el objeto files enviado en la misma es 0. Esto es:

```
1 app.post('/upload', async (req, res) => {
2   // Validando la no existencia de un archivo vacío
3   if (!req.files || Object.keys(req.files).length === 0) {
4     res.send({
5       status: false,
6       message: 'Archivo no subido al servidor',
7       error: "400"
8     });
9   } else { // Se procede con la subida del archivo al servidor
10    let fileRecived = req.files.fileName;
11    let extName = fileRecived.name;
12    console.log(fileRecived);
13    uploadPath = './files/' + extName;
14
15    fileRecived.mv(uploadPath, err => {
16      if (err) {
17        return res.status(500).send({
18          message: err
19        });
20      }
21      return res.status(200).send({
22        message: 'Archivo Subido al Servidor'
23      });
24    });
25  }
26 });
```

Verificamos con Postman al no seleccionar algún archivo:

http://localhost:3000/upload

POST ▼ http://localhost:3000/upload

Params Authorization Headers (8) **Body** ● Pre-request Script Tests Settings

☐ none
 ☒ form-data
 ☐ x-www-form-urlencoded
 ☐ raw
 ☐ binary
 ☐ GraphQL

KEY	VALUE
<input checked="" type="checkbox"/> fileName	Select Files
Key	Value

Body Cookies Headers (6) Test Results 🌐 Stat

Pretty Raw Preview Visualize JSON ▼ ≡

```

1  {
2    "status": false,
3    "message": "Archivo no subido al servidor",
4    "error": "400"
5  }
  
```

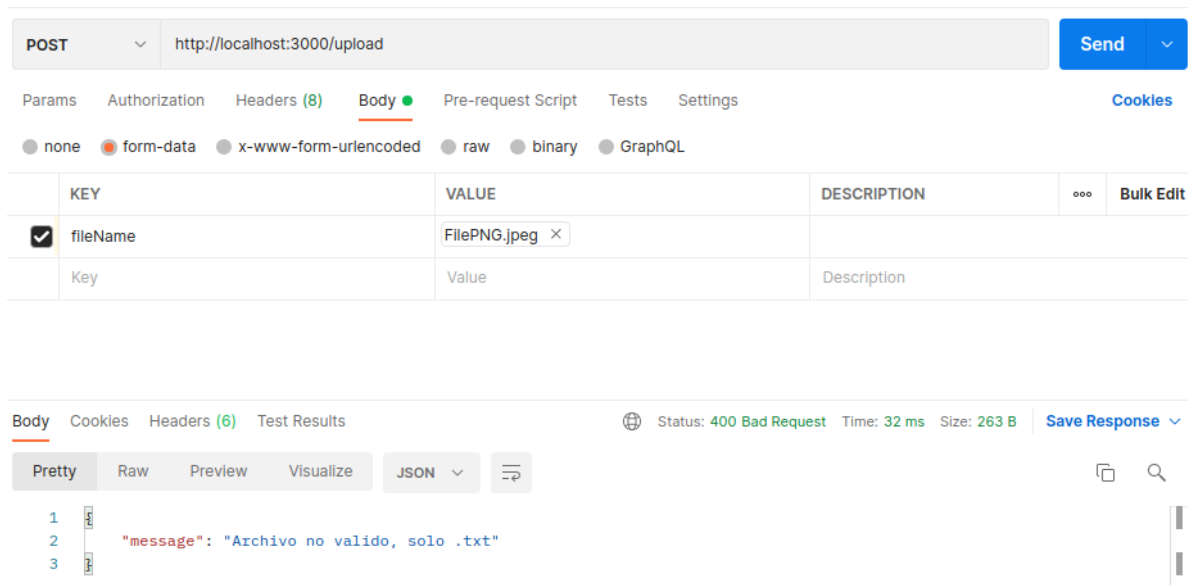
Procedemos a verificar la subida de archivos según el tipo de archivos. En este caso solo vamos a permitir archivos del tipo texto, para ello debemos validar en la condicional que se **"text/plain"**:

```

1 app.post('/upload', async (req, res) => {
2   // Validando la no existencia de un archivo vacío
3   if (!req.files || Object.keys(req.files).length === 0) {
4     res.send({
5       status: false,
6       message: 'Archivo no subido al servidor',
7       error: "400"
8     });
9   } else { // Se procede con la subida del archivo al servidor
10    let fileReceived = req.files.fileName;
11    let extName = fileReceived.name;
12    console.log(fileReceived);
13    uploadPath = './files/' + extName;
14    //Validando que el archivo sea solo texto plano .txt
15    if (fileReceived.mimetype === "text/plain") {
16
17      fileReceived.mv(uploadPath, err => {
18        if (err) {
19          return res.status(500).send({
20            message: err
          }
        )
      }
    }
  }
}
  
```

```
21         });  
22     }  
23     return res.status(200).send({  
24         message: 'Archivo Subido al Servidor'  
25     });  
26 });  
27 } else {  
28     return res.status(400).send({  
29         message: 'Archivo no valido, solo .txt'  
30     });  
31 }  
32 }  
33 }  
34 });
```

Verificamos en el Postman al subir otro tipo de archivos. Por ejemplo .png, y observamos la respuesta:



POST http://localhost:3000/upload Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> fileName	FilePNG.jpeg x			
Key	Value	Description		

Body Cookies Headers (6) Test Results Status: 400 Bad Request Time: 32 ms Size: 263 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {  
2   "message": "Archivo no valido, solo .txt"  
3 }
```

NOTA:

Para una mejor resolución del proyecto, es recomendable tener actualizada la versión de Node.js a la más reciente, y codificar con ES6.