<codoa codo/>

FULL STACK FRONTEND
Clase 30

Node 6

BA

Les damos la bienvenida

Vamos a comenzar a grabar la clase

Clase 29

Node

- Modulo mysql2
  Estructura del proyecto
  Controladores
  Altas
  Modificaciones
  Bajas

Clase 30

Node

- MULTER
  Instalación y Configuración
  diskStorage y filename
  Subiendo archivos
  Probando desde Postman

Clase 31

Node

+ MULTER

Subiendo archivos al servidor

¿Qué es MULTER?

MULTER es un middleware de Node.js para manejar la subida de archivos.
En general se usa con express y facilita la gestión de archivos recibidos en peticiones HTTP.

¿Por qué usar MULTER?

Sencillez: Configuración y uso sencillo

Flexibilidad: Soporta varios tipos de almacenamiento.

Control: Permite configurar de manera detallada el destino y nombre de los archivos.

Instalación y configuración básica:

Para instalar MULTER utilizaremos el gestor de paquetes:

```
npm install multer
```

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

const storage = multer.diskStorage({
    destination: (req, file, cb) => {
        cb(null, 'uploads/');
    },
    filename: (req, file, cb) => {
        cb(null, Date.now() + path.extname(file.originalname));
    }
});

const upload = multer({ storage: storage });
```

Veamos a continuación en más detalle el método .diskStorage

Método .diskStorage (destination)

Es un método en Multer que se utiliza para configurar cómo y dónde se almacenan los archivos subidos. Permite especificar el destino y el nombre de los archivos.

```
const storage = multer.diskStorage({
    destination: (req, file, cb) => {
        cb(null, 'uploads/');
    },
});
```

Dentro del método **.diskStorage** tenemos la función **.destination** que recibe como parámetro una request, el archivo y un callback.

En el callback a su vez recibe se especifica la carpeta destino ‘uploads’.

Método .diskStorage (filename)

Es otro método en Multer que se utiliza para configurar el nombre del archivo que estamos recibiendo. Siempre que recibamos un archivo del front, el nombre del archivo habrá que cambiarlo para asegurarnos que sea un nombre único dentro de nuestra storage.

```
const storage = multer.diskStorage({
    filename: (req, file, cb) => {
        cb(null, Date.now() + path.extname(file.originalname));
    },
});
```

La función **.filename** recibe como parámetro una request, el archivo y un callback.
En la generación del nombre del archivo generamos un nombre único usando la marca de tiempo actual + el nombre original del archivo.

.diskStorage – Ejemplo con diskStorage y filename

En el siguiente ejemplo combinamos la configuración del destino y el nombre del archivo, creando una instancia (upload) de Multer con la configuración personalizada.

```
const multer = require('multer');
const path = require('path');

const storage = multer.diskStorage({
    destination: (req, file, cb) => {
        cb(null, 'uploads/');
    },
    filename: (req, file, cb) => {
        cb(null, Date.now() + path.extname(file.originalname));
    }
});

const upload = multer({ storage: storage });
```

```
// Importa el módulo 'multer', que es un middleware para manejar la subida de archivos en aplicaciones Express.
const multer = require('multer');

// Importa el módulo 'path', que proporciona utilidades para trabajar con rutas de archivos y directorios.
const path = require('path');

// Configura el almacenamiento de archivos con 'diskStorage'.
const storage = multer.diskStorage({
    // 'destination' especifica la carpeta donde se guardarán los archivos subidos.
    // 'cb' es el callback que se llama después de determinar el destino.
    destination: (req, file, cb) => {
        // El primer argumento de 'cb' es para el error (null si no hay error).
        // El segundo argumento es la carpeta donde se guardará el archivo.
        cb(null, 'uploads/');
    },
    // 'filename' especifica cómo se nombrarán los archivos subidos.
    // 'cb' es el callback que se llama después de determinar el nombre del archivo.
    filename: (req, file, cb) => {
        // El primer argumento de 'cb' es para el error (null si no hay error).
        // El segundo argumento es el nombre del archivo.
        // 'Date.now()' se usa para asegurar que cada archivo tenga un nombre único basado en la marca de tiempo actual.
        // 'path.extname(file.originalname)' obtiene la extensión del archivo original.
        cb(null, Date.now() + path.extname(file.originalname));
    }
});

// Crea una instancia de 'multer' con la configuración de almacenamiento definida.
// 'storage' es el almacenamiento configurado que hemos definido previamente.
const upload = multer({ storage: storage });
```

Uso de rutas: Subiendo un solo archivo

Una de las principales cuestiones al subir un archivo, es definir dónde guardaremos el archivo que estamos recibiendo.

.single: Este es un ejemplo cuando estamos esperando un solo archivo.

```
// Define una ruta POST para subir un archivo. La URL de la ruta es '/upload'.
app.post('/upload',

    // Middleware de Multer para manejar la subida de un solo archivo.
    // 'upload.single('archivo')' indica que se subirá un solo archivo con el nombre de campo 'archivo'.
    upload.single('archivo'),

    // Callback que se ejecuta después de que Multer procesa el archivo subido.
    (req, res) => {
        // Envía una respuesta al cliente indicando que el archivo se ha subido con éxito.
        res.send('Archivo subido con éxito');
    }
);
```

Uso de rutas: Subiendo multiples archivos

Una de las principales cuestiones al subir un archivo, es definir dónde guardaremos el archivo que estamos recibiendo.

.array: Este es un ejemplo cuando esperamos más de un archivo.

```
// Define una ruta POST para subir múltiples archivos. La URL de la ruta es '/uploads'.
app.post('/uploads',

    // Middleware de Multer para manejar la subida de múltiples archivos.
    // 'upload.array('archivos', 10)' indica que se pueden subir hasta 10 archivos con el nombre de campo 'archivos
    upload.array('archivos', 10),

    // Callback que se ejecuta después de que Multer procesa los archivos subidos.
    (req, res) => {
        // Envía una respuesta al cliente indicando que los archivos se han subido con éxito.
        res.send('Archivos subidos con éxito');
    }
);
```

Validación de los archivos recibidos

Una de las principales cuestiones a recibir archivos de manera externa es validar que estamos recibiendo el tipo de archivo y el tamaño apropiados.

No podemos confiar en que el usuario, o el frontend envíen exactamente lo que estamos esperando.

Por ese motivo tenemos dentro de Multer las opciones **fileFilter** y **limits.**

fileFilter: Nos permite filtrar por tipos de archivo. En el caso de las imágenes podemos definir que queremos solo archivos .jpg o una combinación de .jpg, .png y .jpeg

limits: Establece el tamaño máximo que puede tener el archivo.

Validación de los archivos recibidos

```
// Configura la instancia de Multer con opciones adicionales.
const upload = multer({
    // Utiliza la configuración de almacenamiento previamente definida.
    storage: storage,

    // Función para filtrar los archivos según su tipo.
    fileFilter: (req, file, cb) => {
        // Define los tipos de archivos permitidos usando una expresión regular.
        const filetypes = /jpeg|jpg|png/;

        // Verifica si el tipo MIME del archivo es uno de los permitidos.
        const mimetype = filetypes.test(file.mimetype);

        // Verifica si la extensión del archivo es una de las permitidas.
        const extname = filetypes.test(path.extname(file.originalname).toLowerCase());

        // Si ambos, tipo MIME y extensión del archivo, son permitidos, acepta el archivo.
        if (mimetype && extname) {
            return cb(null, true);
        }

        // Si el archivo no es permitido, devuelve un error.
        cb('Error: Tipo de archivo no soportado');
    },

    // Define límites para el archivo, como el tamaño máximo en bytes.
    limits: { fileSize: 1000000 } // 1MB
});
```

MIME Type, significa Multipurpose Internet Mail Extensions Type. Indica el tipo de contenido y formato de un archivo. Asegura que los archivos sean reconocidos y manejados correctamente, mejorando la seguridad y compatibilidad.

Su estructura es: type/subtype.

Utilizando HTTP se especifica en el encabezado **Content-Type** para definir el tipo de contenido de las solicitudes y respuestas.

Un ejemplo aplicando lo visto:

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

const app = express();
const port = 3000;

const storage = multer.diskStorage({
    destination: (req, file, cb) => {
        cb(null, 'uploads/');
    },
    filename: (req, file, cb) => {
        cb(null, Date.now() + path.extname(file.originalname));
    }
});

const upload = multer({ storage: storage });

app.post('/upload', upload.single('archivo'), (req, res) => {
    res.send('Archivo subido con éxito');
});

app.listen(port, () => {
    console.log(`Servidor corriendo en http://localhost:${port}`);
});
```

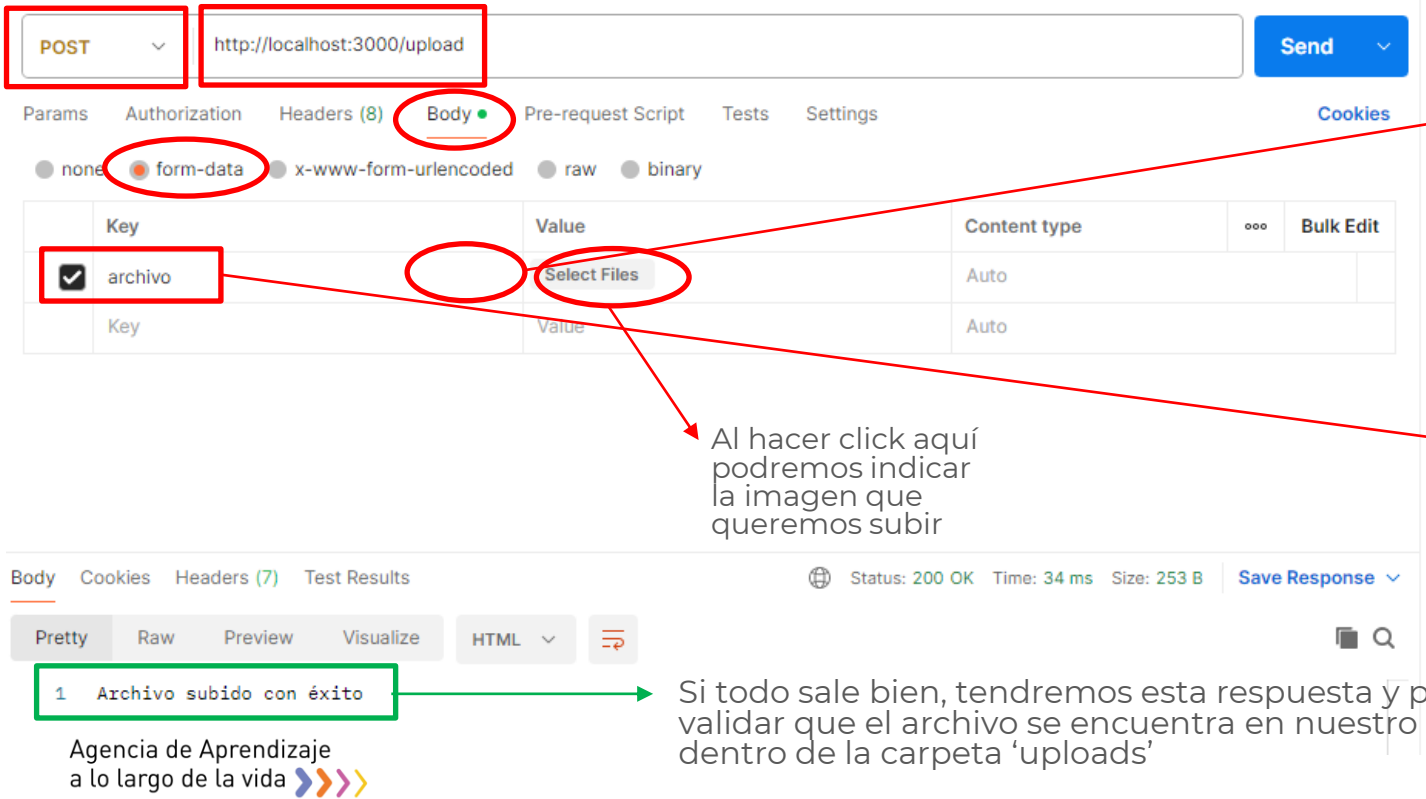
Con este código podemos probar la potencia de multer, que nos permitirá recibir una imagen desde un frontend o bien desde Postman.

Esta carpeta debe existir en nuestro proyecto.

El nombre de la key que utilizaremos en el form-data

La ruta que utilizaremos en el endpoint

Preparando Postman para probar:



Haciendo click en este espacio podremos definir si queremos enviar un dato de tipo 'text' o de tipo 'file'. Debemos elegir 'file' par a que habilie en Value la carga del archivo.

Al hacer click aquí podremos indicar la imagen que queremos subir

La key debe tener el nombre que definimos en el servidor.

Si todo sale bien, tendremos esta respuesta y podremos validar que el archivo se encuentra en nuestro proyecto dentro de la carpeta 'uploads'

Agencia de Aprendizaje a lo largo de la vida

No te olvides de dar el presente

Recordá:

- Revisar la Cartelera de Novedades.
- Hacer tus consultas en el Foro.
- **Realizá los ejercicios obligatorios.**

Todo en el Aula Virtual.