

Agencia de Aprendizaje a lo largo de la vida

FULL STACK FRONTEND Clase 30

Node 6





Les damos la bienvenida

Vamos a comenzar a grabar la clase





Clase de apoyo Viernes 28/06 "Autentificación web con Google"

Agencia de Habilidades
para el Futuro





Link a la clase de refuerzo

Información para unirse con Google Meet de 15 a 17 Enlace de la videollamada: https://meet.google.com/sjw-gkny-wam







Clase 29 Clase 30

Node

Modulo mysql2
 Estructura del proyecto
 Controladores
 Altas
 Modificaciones
 Bajas

Node

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

Node













¿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:

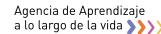
npm install multer

```
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');

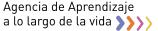
const storage = multer.diskStorage({
    destination: (req, file, cb) => {
        cb(null, 'uploads/');
    },
    filename: (req, file, cb) => {
        cb(null, Date.now() + path.extname(file.originalname));
    }
});

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





```
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 });
```







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

```
const upload = multer({
   storage: storage,
   fileFilter: (req, file, cb) => {
       const filetypes = /jpeg jpg png/;
       const mimetype = filetypes.test(file.mimetype);
       const extname = filetypes.test(path.extname(file.originalname).toLowerCase());
       if (mimetype && extname) {
           return cb(null, true);
       cb('Error: Tipo de archivo no soportado');
   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.

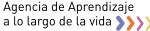


Un ejemplo aplicando lo visto:

```
const express = require('express');
const multer = require('multer');
const path = require('path');
const app = express();
const port = 3000;
const storage = multer.diskStorage({
    destination: (reg. 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.singl(('archivo')) (req, res) => {
    res.send('Archivo subido con exito');
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.

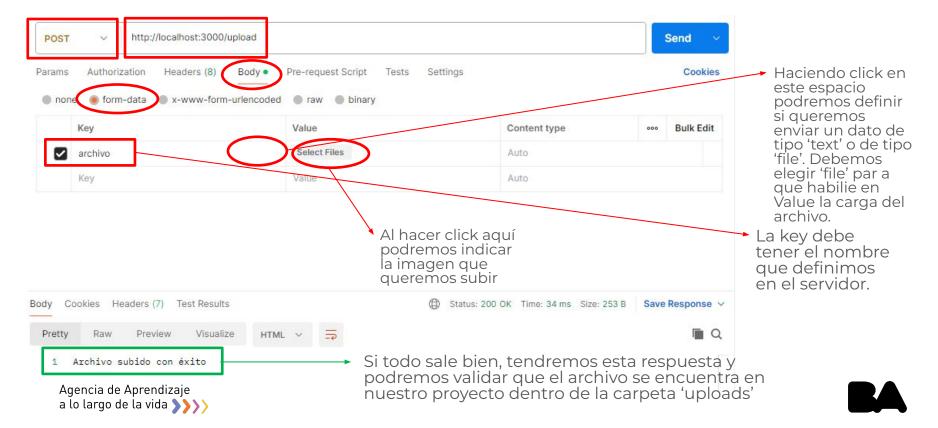
- Esta carpeta debe existir en nuestro proyecto.
- El nombre de la key que utilizaremos en el form-data
- La ruta que utilizaremos en el endpoint







Preparando Postman para probar:







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.