

Agencia de
Aprendizaje
a lo largo
de la vida

Desarrollo Fullstack





Clase 34

Node JS

- ▶ Capa de Vistas
- ▶ Motores de Plantillas
- ▶ Sintaxis EJS

Clase 35

Node JS

- ▶ Multer
- ▶ Express Validator

NODE JS

Multer + Express Validator



Multer

Es un middleware muy popular en aplicaciones NODE, utilizado para manejar la subida de archivos al servidor.

Nos permite manejar data enviada a través del `enctype="multipart/form-data"`, formato utilizado para enviar archivos desde un formulario.

Instalación

Corremos el comando: `npm install multer`

Una vez instalado solo debemos invocar a la librería:

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

Multer posee diversos para trabajar con nuestros archivos pero nosotros utilizaremos: `.diskStorage()`

Configuración

Para configurar nuestra instancia de **multer** necesitamos definir **2** propiedades: **destination** y **filename**.

destination

Lugar donde alojaremos los archivos subidos.

filename

Nombre que le asignaremos a esos archivos cuando sean subidos.

Veamos cómo es en el código...

Implementación

```
const storage = multer.diskStorage({
  destination: (req, file, cb) => cb(null, path.resolve(__dirname, '../..public/img')),
  filename: (req, file, cb) => cb(null, Date.now() + '_' + file.originalname)
});

const uploadFile = multer({storage});
```

storage: guarda la configuración de multer.

uploadFile: crea una instancia de multer y le asigna la configuración creada.

Implementación

Ahora, nuestra variable **uploadFile** tiene distintos métodos para subir archivos.

```
const uploadFile = multer({storage});  
  
uploadFile.  
  any (method) mul  
  array  
  fields  
  none  
  single
```

single('fieldname'): se utiliza cuando solo se espera un solo archivo. **fieldname** es el name="" del input

array('fieldname', max): se utiliza cuando se esperan múltiples archivos en un mismo input. **max** es el máximo de archivos permitido.

fields(fields): se utiliza cuando se esperan múltiples inputs del **type="file"**. **fields** es un array de conjuntos **{'fieldname', max}**.


El próximo paso es capturar los archivos en las request...

Implementación

Al ser un **middleware** a nivel de ruta, debemos ejecutar multer entre la ruta y el controlador que deseemos.

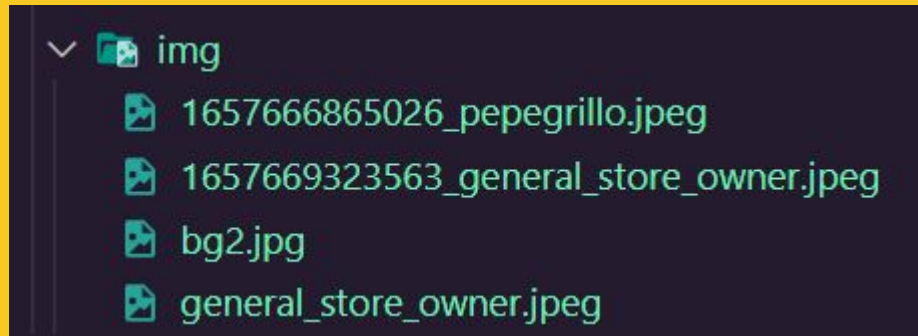
```
<!-- HTML -->
<form action="/admin/edit/20?_method=PUT" method="POST" enctype="multipart/form-data">
  <input type="file" name="image" id="image">
  <button type="submit">Editar</button>
</form>

/* Archivo de Rutas */
router.put('/edit/:id', uploadFile.single('image'), adminController.save);
```



Aquí capturamos y guardamos los archivos, antes de ir al controlador.

De esta manera
podemos guardar
archivos enviados
desde una request.



Express Validator

Express Validator es una dependencia utilizada como middleware a nivel ruta, que se utiliza para validar y sanitizar datos de entrada.



Express validator

Con esta librería podremos:

Validar datos recibidos desde el formulario

Manejar errores de validación y personalizar nuestros mensajes de error al enviar datos.

Sanitizar datos para eliminar caracteres no deseados y prevenir ataques de XSS (Cross Side Scripting) y SQL Injections.

Integración sencilla con Express.js

Instalación

Corremos el comando: `npm install express-validator`

Luego requerimos la dependencia donde necesitemos utilizarla:

```
const { body, validationResult, sanitizeBody } = require('express-validator');
```

Como vemos, **express-validator** posee diversos métodos a utilizar, en función de lo que necesitemos validar.

Implementación

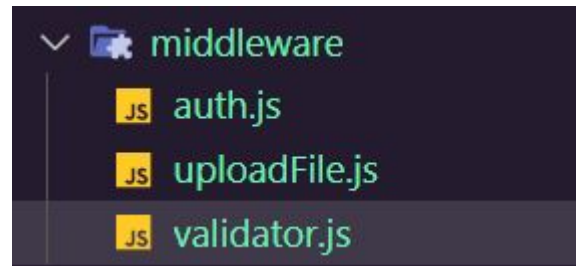
```
const { validationResult } = require('express-validator');

// Middleware de validación

const validateInput = (req, res, next) => {
  const errors = validationResult(req); // Extraemos los errores
  if (!errors.isEmpty()) {
    // Si hay errores, los devolvemos al cliente
    return res.status(400).json({ errors: errors.array() });
  }
  next(); // Caso contrario continuamos al controlador
};

module.exports = validateInput;
```

Creamos nuestro middleware y lo exportamos.



Implementación

```
const { body } = require('express-validator');
const validateInput = require('../middleware/validator');

const loginValidations = [
  body('email')
    .isEmail()
    .withMessage('Ingresa una dirección de correo electrónico válida'),
  body('password')
    .isLength({ min: 6 })
    .withMessage('La contraseña debe tener al menos 6 caracteres')
];

// Ruta de ejemplo con el middleware de validación
app.post('/login', loginValidations, validateInput, loginController.checkLogin);
```

Primero pasamos las validaciones en **loginValidations**, luego nuestro **middleware** maneja el **resultado** de las validaciones en **validateInput** donde si son es **correcto** continúa con el controlador o envía los **errores** al cliente.

Validaciones

Existen diversos tipos de **validaciones** y la librería ofrece distintas formas de realizarlas como mediante un **schema**:

```
checkSchema({  
  email: {  
    errorMessage: 'Invalid username',  
    isEmail: true,  
  },  
  password: {  
    isLength: {  
      options: { min: 8 },  
      errorMessage: 'Password should be at least 8 chars',  
    },  
  },  
});
```

Podemos acceder a la documentación para ver la lista completa de validaciones que se pueden realizar:

<https://express-validator.github.io/docs/api/validation-chain>

No te olvides de dar el presente

Recordá:

- **Revisar la Cartelera de Novedades.**
- **Hacer tus consultas en el Foro.**

Todo en el Aula Virtual.

Gracias