Express Validator
Introducción

DigitalHouse >
Coding School

“Express Validator nos permite manejar de manera relativamente sencilla, todas las validaciones de nuestros formularios desde el lado del back-end.”

Express Validator paso a paso

Express Validator nos permitirá validar que la información que nos llega desde los formularios sea la que esperamos.

Los pasos que debemos seguir para implementarlo son:

1. **Instalar Express Validator.**
2. **Crear un array con las validaciones de cada formulario.**
3. **Agregarlo como middleware de la ruta que procesa el formulario.**
4. **Verificar si hubo errores en la validación desde el controlador.**
5. **Enviar los errores a las vista.**

¡Vamos entonces a ver los primeros tres pasos!

Instalando Express Validator

Lo primero que debemos hacer **es instalar el paquete** en nuestra aplicación, es decir, dentro de la carpeta donde tenemos el proyecto.

Para ello ejecutamos el siguiente comando:

```
>_  npm i express-validator
```

Nuestro formulario

Antes de empezar con las validaciones, es importante tener en cuenta cómo están armados nuestros formularios. A la hora de escribir las validaciones, tomaremos como referencia la propiedad **name** de cada campo.

{}

```
<form action="/registro" method="post">
     <label for="name">Nombre:</label>
     <input type="text" name="name" id="name">

     <label for="email">Correo electrónico:</label>
     <input type="email" name="email" id="email">

     <label for="password">Contraseña:</label>
     <input type="password" name="password" id="password">

     <button type="submit">Registrarse</button>
</form>
```

Las validaciones

Ahora que tenemos el módulo instalado, vamos a requerirlo donde vayamos a realizar las validaciones. Podemos hacerlo directamente sobre el archivo de rutas o crear nuestras validaciones en un archivo aparte.

En cualquiera de los casos, el primer paso será requerir el módulo y, haciendo uso de la desestructuración, pedir el método **check**.

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

El segundo paso será crear una variable donde almacenaremos el conjunto de validaciones que realizaremos sobre el formulario.

```
{}  let validateRegister = [] // validaciones aquí
```

El método check()

El método **check()** nos permite agregar validaciones para cualquiera de los campos del formulario. Como parámetro recibe el nombre del campo a validar. Si por ejemplo queremos validar el campo **name**, el método quedaría así:

```
{}  const validateRegister = [ check('name') ]
```

Suponiendo que quisiéramos validar que el campo no esté vacío, sobre el método anterior, ejecutamos el método **notEmpty()** de la siguiente manera:

```
{}  const validateRegister = [
        check('name').notEmpty()
    ]
```

Tipos de validaciones

Existen varios tipos de validaciones, veamos las más comunes:

```
{}
check('campo')
    .notEmpty() // Verifica que el campo no esté vacío
    .isLength({ min: 5, max: 10 }) // Verifica la longitud de los datos
    .isEmail() // Verifica que sea un email válido
    .isInt() // Verifica que sea un número entero
```

Como pueden ver, podemos tener más de una validación para el mismo campo. Si ese es el caso, simplemente ejecutamos un método seguido del otro.

La lista completa de validaciones puede verse aquí 📌

Mensajes de error

Además de las validaciones, Express Validator nos permite definir el mensaje que recibirá el usuario por cada validación que falle.

Para implementar los mensajes, utilizamos el método **withMessage()** a continuación de cada validación.

```
{}
check('name')
    .notEmpty().withMessage('Debes completar el nombre')
    .isLength({ min: 5 }).withMessage('El nombre debe tener al menos 5 caracteres')
```

Cortando la cadena de validación

En algunos casos vamos a querer cortar la validación, ya que si por ejemplo un campo está vacío, no tiene sentido verificar si es un e-mail válido.

Si no cortamos la validación, el usuario recibirá todos los errores juntos en lugar de solo el que corresponda.

Para esos casos, podemos implementar el método **bail()**.

```
{}
check('email')
    .notEmpty().withMessage('Debes completar el email').bail()
    // En caso de que la primera validación falle,
    // las siguientes no se ejecutan para ese campo.
    .isEmail().withMessage('Debes completar un email válido')
```

El array de validaciones completo

En resumen, cuando terminemos de escribir nuestras validaciones, tendremos un array, con un elemento por campo, con todas sus validaciones.

```
{}
const validateRegister = [
    check('name')
        .notEmpty().withMessage('Debes completar el nombre').bail()
        .isLength({ min: 5 }).withMessage('El nombre debe ser más largo'),
    check('email')
        .notEmpty().withMessage('Debes completar el email').bail()
        .isEmail().withMessage('Debes completar un email válido'),
    check('password')
        .notEmpty().withMessage('Debes completar la contraseña').bail()
        .isLength({ min: 8 }).withMessage('La contraseña debe ser más larga')
]
```

Agregando las validaciones en las rutas

Terminadas nuestras validaciones es hora de implementarlas y para eso las agregaremos en la ruta que procese el formulario que queremos validar.

Este middleware, se ubica entre la ruta y la acción del controlador.

{}

```
const validateRegister = [ ... ];

// Procesamiento del formulario de creación
router.post('/', validateRegister, userController.processRegister);
```


DigitalHouse>
Coding School