# Express Validator Avanzado

DigitalHouse >
Coding School

“Express Validator nos permite generar mensajes por cada error de validación, que podremos enviar a la vista para que los vea el usuario.”

# Express Validator paso a paso

Los pasos anteriores se encargaron de que ocurriera el proceso de validación del formulario. Nos toca ahora trabajar con el resultado de esa validación.

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 últimos dos pasos!

# Las validaciones en el controlador

Nuevamente, el primer paso será requerir el módulo y, haciendo uso de la desestructuración, pedir el método **validationResult**.

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

El segundo paso ocurrirá dentro del método del controlador que se encarga de procesar el formulario. Allí guardaremos, en la variable errors, la ejecución del método **validationResult**, pasándole como parámetro el objeto request.

```
{}  let errors = validationResult(req);
```

# El método isEmpty()

El método **isEmpty()** nos permite saber si hay errores de validación o no.

Si no hay errores, podremos seguir sin inconvenientes con la acción que deba realizar ese controlador, por ejemplo, crear un nuevo usuario.

En caso contrario, volveremos al formulario con los errores para el usuario.

```
{}
    register: (req, res) => {
        let errors = validationResult(req);

        if (errors.isEmpty()) {
            // No hay errores, seguimos adelante
        } else {
            // Si hay errores, volvemos al formulario con los mensajes
        }
    },
```

# El método mapped()

El método **mapped()** nos permite enviar los errores a la vista como un objeto. Ese objeto contendrá una propiedad con el primer error de cada campo.

Para enviar los errores a la vista, simplemente los agregamos como segundo parámetro al método **render()**.

Es importante enviar también los contenidos de **req.body**, ya que queremos preservar los datos completados por el usuario al volver al formulario.

```
{}
```

```
if (errors.isEmpty()) {
    // No hay errores, seguimos adelante
} else {
    // Hay errores, volvemos al formulario con los mensajes
    res.render('register', { errors: errors.mapped(), old: req.body });
}
```

# El objeto de errores

Por cada campo con error, el objeto tendrá una propiedad cuyo nombre será igual al atributo name del campo.

Cada error contendrá: el mensaje (**msg**), el nombre del campo (**param**), el valor ingresado por el usuario (**value**) y de donde vino (**body** para formularios).

{}

```
{
   email: {
       msg: 'Debes completar un email válido',
       param: 'email',
       value: 'unEmail',
       location: 'body'
   },
   password: {
       msg: 'La contraseña debe ser más larga',
       param: 'password',
       value: '1234',
       location: 'body'
   }
}
```

# Mostrando los errores en la vista

Haciendo uso de EJS, podremos preguntar si un campo determinado tiene errores. Si ese es el caso, podremos mostrar el mensaje de error.

Es importante tener en cuenta que la primera vez que se cargue el formulario no habrá errores, y por lo tanto esa variable estará vacía. Para evitar problemas, siempre debemos preguntar si la variable de errores existe antes de intentar mostrar un error.

El código podría verse de la siguiente manera:

```
{}
```

```
<label for="email">Correo electrónico:</label>
<input type="email" name="email" id="email">
<% if (locals.errors && errors.name) { %>
    <p class="feedback"><%= errors.name %></p>
<% } %>
```

# Agregando los datos anteriores al formulario

Otro punto importante es que si el usuario ya completó el formulario, pero puso información inválida en algún campo, no vamos a querer que complete todo nuevamente.

Por esa razón, en el paso anterior volvimos a enviar los datos del formulario original en el objeto **old**.

Nuevamente con EJS podemos cargar ese valor en cada campo que corresponda.

```
{}
        <label for="email">Correo electrónico:</label>
        <input type="email" name="email" id="email"
               value="<%= locals.old && old.email %>">
```


DigitalHouse>
Coding School