Programación Web Full Stack

Validaciones con Express Validator

Guía paso a paso

Punto de partida

Sin duda las validaciones son esenciales a la hora de que un sitio funcione correctamente y de que la información que llegue al servidor sea la que esperamos.

Existen dos tipos de validaciones: las del front-end y las del back-end. Estas últimas son las más importantes porque son las que se ejecutarán siempre, y constituyen nuestro último punto de verificación antes de que los datos se guarden definitivamente en nuestras bases de datos. En esta guía nos enfocaremos justamente en las validaciones de back-end.

Para poder seguir este paso a paso el proyecto deberá contar con lo siguiente:

1. Un proyecto con Node.js y Express instalado y funcionando.
2. Una estructura MVC donde existan ruteadores y controladores.
3. Vistas con EJS o cualquier otro motor de plantillas.
4. Un formulario que ya esté funcionando y recibiendo datos.

Para este ejemplo vamos a imaginar que estamos validando el registro de un usuario.

¡Hora de empezar! 🤓👌

 Paso 1: la instalación

Para validar vamos a utilizar **Express Validator**. Para instalarlo utilizaremos el siguiente comando:

```
npm i express-validator
```

 Paso 2: la implementación en las rutas

Express Validator es un middleware de ruta, lo cual quiere decir que tendremos que implementarlo para cada ruta donde haya un formulario por validar.

Asumiendo que estamos validando la creación de un nuevo usuario, en el archivo de rutas de usuarios vamos requerir el módulo que instalamos en el paso previo.

Como en este caso no queremos toda la funcionalidad de Expess Validator, utilizaremos desestructuración para solamente pedir el método **check**.

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

Pensando en que nuestro formulario tiene los siguientes campos:

- Nombre → first_name
- Apellido → last_name
- Email → email
- Clave → password

En la práctica podría verse de la siguiente manera:

```
<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>
```

Vamos a comenzar a escribir algunas validaciones utilizando el método **check**. Para eso podemos definir una variable que almacenará un array de validaciones:

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

Los elementos del array consistirán en utilizar el método **check** por cada campo que queramos validar. Como parámetro, check recibirá el nombre del campo a validar, es decir, literalmente el valor de su atributo name.

```
 check('first_name')
```

Seguido de eso, agregaremos los métodos de validaciones que queramos utilizar, **notEmpty**, **isEmail**, **isLength**, etc. El listado completo puede verse aquí.

```
 check('first_name').notEmpty()
```

Opcionalmente, podemos configurar un mensaje de error utilizando el método **withMessage()**:

```
 check('first_name').notEmpty().withMessage('Debes completar el nombre')
```

Si queremos tener más de una validación por campo, es cuestión de seguir agregando métodos de validación uno después del otro.

```
check('first_name')
    .notEmpty().withMessage('...')
    .isLength({ min: 5 }).withMessage('...'),
```

Por último, el método **bail()** nos permitirá cortar la cadena de validación en cualquier momento. Si por ejemplo un campo es obligatorio, una vez que detectamos que está vacío, no tiene sentido continuar con el resto de validaciones que pueda haber para ese campo.

```
check('first_name')
    .notEmpty().withMessage('...').bail()
    .isLength({ min: 5 }).withMessage('...'),
```

Vamos a ver cómo podría verse todo junto:

```
const validateRegister = [
     check('first_name')
          .notEmpty().withMessage('Debes completar el nombre').bail()
          .isLength({ min: 5 }).withMessage('El nombre debe tener al menos 5
caracteres'),
      check('last_name')
          .notEmpty().withMessage('Debes completar el apellido').bail()
          .isLength({ min: 5 }).withMessage('El apellido debe tener al menos
5 caracteres'),
      check(email)
          .notEmpty().withMessage('Debes completar el email).bail()
          .isEmail().withMessage('Debes ingresar un email válido'),
      check('password')
          .notEmpty().withMessage('Debes completar la contraseña').bail()
          .isLength({ min: 5 }).withMessage('La contraseña debe tener al
menos 5 caracteres');
```

Una vez que terminemos de escribir nuestro array de validaciones, quedará implementarlo en la ruta que procese el formulario que queremos validar.

Recordemos que este tipo de middlewares debe ir siempre entre la ruta y la acción del controlador como se muestra en el ejemplo:

```
router.post('/register, validateRegister, controller.processRegister);
```

Paso 3: la implementación en los controladores

Lo que hicimos hasta el momento se encarga de validar los formularios y enviarnos el resultado de dicha validación. Nos queda entonces verificar el resultado y construir la respuesta tanto para el caso de que la información enviada sea la correcta como para el caso de que sea incorrecta.

Nos encontramos en el controlador y nuevamente haremos uso de la desestructuración para pedir un método específico de Express Validator. En este caso se trata de **validationResult**.

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

Para implementar **validationResult**, deberemos ejecutarlo pasándole como parámetro el request y guardar los resultados en una variable. Al igual que con la ruta, esto deberemos hacerlo en el método que procesa el formulario.

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

validationResult() nos provee un método llamado **isEmpty()** que nos permitirá determinar si hay o no errores de validación. Utilizando este método podremos escribir lógica para ambos casos.

```
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.
    }
},
```

En caso de que haya errores, vamos a querer que estos lleguen a la vista. La manera de lograr eso es volviendo a renderizar el formulario, pero esta vez enviándole los errores de validación. Para eso vamos a utilizar el método **mapped()** para obtener un objeto literal con los errores.

Lo segundo que vamos a querer hacer es enviarle el resto de los datos completados por el usuario en el formulario. Recordemos que estos datos llegan en la propiedad **body** del objeto **request**.

Nuestro código podría verse algo así:

```
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 });
}
```

Paso 4: la implementación en las vistas

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>
    <% } %>
```

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**. 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 %>">
```

Conclusión

Las validaciones son imprescindibles si queremos que nuestro sitio funcione correctamente y evitar problemas cuando nuestros usuarios lo utilicen.

Si bien el uso de middlewares de validación lleva un poco de trabajo, nos asegura que toda la información que llegue a nuestras bases de datos será la correcta. También nos permite avisarle al usuario, con detalle, en caso de que alguno de los datos sea incorrecto o no tenga el formato que esperamos.

Con esto ya estamos un paso más cerca de tener un sitio 100% funcional.

¡Hasta la próxima!