Node.js: come usare Express Validator in ExpressJS

ExpressJS può contare su un modulo NPM specifico per effettuare la validazione dei dati lato server.

express-validator è un middleware che effettua la validazione dei dati prima che venga eseguita la funzione di callback che gestisce una route. Per le funzionalità di validazione e filtraggio dei dati è interamente basato sul modulo validator.js da cui eredita i metodi principali.

In una tipica richiesta POST di un form, questo modulo definisce le sue regole come array di middleware e quindi rende disponibile i risultati aggiungendo un array di oggetti all'oggetto `request`.

```
'use strict';

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

app.post('/contact',
[
  body('email').isEmail(),
  body('subject').isLength({ min: 5 }),
  body('message').isLength({ min: 100 }),

], (req, res) => {
    const validationErrors = validationResult(req);

    if (!validationErrors.isEmpty()) {
        return res.status(400).render('contact', {
errors: validationErrors.array() } );
```

```
      }

      //...
});
```

`body('nomecampo')` verifica i parametri dell'oggetto `request.body` che viene popolato durante l'invio del form come richiesta POST. `nomecampo` corrisponde al nome dell'attributo `name` in un form HTML e quindi al nome del parametro della richiesta POST.

Verifichiamo prima se si siano errori di validazione tramite il metodo `isEmpty()`. Quindi l'output ottenuto trasformando in array i risultati della funzione `validationResult()` è il seguente.

```
[
    {
      location: 'body',
      msg: 'Invalid value',
      param: 'email'
    },
    {
      location: 'body',
      msg: 'Invalid value',
      param: 'subject'
    },
    {
      location: 'body',
      msg: 'Invalid value',
      param: 'message'
    }
]
```

I messaggi di errore (proprietà `msg`) possono essere personalizzati tramite il metodo `withMessage()`.

```
'use strict';

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

app.post('/contact',
[
  body('email').isEmail().withMessage('Invalid e-
mail.'),
  body('subject').isLength({ min: 5
}).withMessage('Required field.'),
  body('message').isLength({ min: 100
}).withMessage('Required field.'),

], (req, res) => {
    const validationErrors = validationResult(req);

    if (!validationErrors.isEmpty()) {
        return res.status(400).render('contact', {
errors: validationErrors.array() } );
      }

      //...
});
```

Questo modulo supporta anche funzioni di validazione personalizzate tramite il metodo `custom()`. Questo metodo deve sollevare un errore o rigettare una Promise se al suo interno si sta usando l'approccio asincrono qualora la verifica abbia esito negativo.

```
'use strict';
```

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

app.post('/contact',
[
  body('email').isEmail(),
  body('subject').isLength({ min: 5 }),
  body('message').isLength({ min: 100 }).custom(value
=> {
     if(/<|>/g.test(value)) {
         throw new Error('Invalid characters
found.');
     }
     return true; // Nessun errore
  }),

], (req, res) => {
    const validationErrors = validationResult(req);

    if (!validationErrors.isEmpty()) {
        return res.status(400).render('contact', {
errors: validationErrors.array() } );
      }

      //...
});
```

Come possiamo visualizzare i messaggi di errore nelle view? Esistono sostanzialmente due approcci. Il primo mostra la lista degli errori all'inizio o alla fine del form. Ad esempio, usando EJS come template engine:

```
<% if(errors.length > 0) { %>
    <% errors.forEach(err => { %>
```

```
        <div class="alert alert-danger"><%= err.msg
%></div>
    <% }); %>
<% } %>

<form action="/contact" method="post" id="contact-
form">

</form>
```

Il secondo approccio mostra ciascun errore vicino al rispettivo campo.

```
<% const hasErrors = errors.length > 0; %>

<form action="/contact" method="post" id="contact-
form">
    <div class="form-group">
        <label for="email">E-mail</label>
        <input type="email" name="email" id="email"
class="form-control">

        <% if(hasErrors) {
           const emailErr = errors.find(e => e.param
=== 'email');
           if(emailErr) {
        %>

        <div class="alert alert-danger"><%=
emailErr.msg %></div>

        <% }

          } %>
    </div>
```

```
    <div class="form-group">
        <label for="subject">Subject</label>
        <input type="text" name="subject"
id="subject" class="form-control">

        <% if(hasErrors) {
           const subjErr = errors.find(e => e.param
=== 'subject');
           if(subjErr) {
        %>

        <div class="alert alert-danger"><%=
subjErr.msg %></div>

        <% }

          } %>
    </div>

    <div class="form-group">
        <label for="message">Message</label>
        <textarea name="message" id="message"
class="form-control">

        <% if(hasErrors) {
           const msgErr = errors.find(e => e.param
=== 'message');
           if(msgErr) {
        %>

        <div class="alert alert-danger"><%=
msgErr.msg %></div>

        <% }
```

```
            } %>
    </div>

        <p><input type="submit" value="Send"
class="btn btn-primary"></p>
</form>
```

In definitiva questo modulo si rivela essere estremamente utile per automatizzare e semplificare il processo di validazione dei dati in un'applicazione in ExpressJS.

Riferimenti

Documentazione