Node.js: servire XML nelle view di ExpressJS

In questo articolo vedremo come servire XML nelle view di ExpressJS quando la nostra applicazione deve essere interoperabile.

Non tutti i linguaggi di programmazione usati sul Web dispongono di un supporto nativo a JSON. Java ad esempio è un linguaggio che storicamente supporta nativamente XML ma non JSON. Se la nostra applicazione mira ad essere interoperabile, dobbiamo fornire un'alternativa al formato JSON.

Nella logica di ExpressJS non cambia molto: il metodo `response.render()` genera per impostazione predefinita un documento HTML. Se vogliamo cambiare questo comportamento, dobbiamo impostare l'header HTTP `Content-Type` su `text/xml` e creare un template XML con il nostro template engine di riferimento (ad esempio EJS).

Supponiamo di voler creare un feed con gli ultimi dieci prodotti di un e-commerce. Creiamo quindi una route specifica:

```
'use strict';

const express = require('express');
const router = express.Router();
const Product = require('../models/product');


router.get('/products.xml', async (req, res, next) =>
{
    try {
        const products = await Product.find().sort({
```

```
added: -1 }).limit(10);
        res.setHeader( 'Content-Type', 'text/xml');
        res.render('products', { products });
    } catch(err) {
        res.sendStatus(500);
    }
});

module.exports = router;
```

Il fatto che la route termini con `.xml` è puramente indicativo e viene qui usato solo per rendere chiaro attraverso l'URL che restituiremo un documento XML.

Dobbiamo creare quindi la view `products` generando la struttura del documento XML in modo dinamico a partire dall'array di documenti restituito dalla query al database.

Il modo più semplice è quello di assegnare ai nomi degli elementi XML lo stesso nome delle proprietà del modello del database.

```
<?xml version="1.0" encoding="UTF-8"?>

<products>
    <% products.forEach(product => { %>
    <product>
        <tags>
            <% if(product.tags.length > 0 ) {
                product.tags.forEach(tag => { %>
                <tag><%= tag %></tag>
            <% });
            } %>
        </tags>
        <price><%= product.price %></price>
```

```
        <name><%= product.name %></name>
        <description><![CDATA[<%= product.description
%>]]></description>
        <slug><%= product.slug %></slug>
        <added><%= product.added %></added>
        <manufacturer><%= product.manufacturer %>
</manufacturer>
        <itemType><%= product.itemType %></itemType>
        <productImg><%= product.productImg %>
</productImg>
    </product>
    <% }); %>
</products>
```

L'unico problema che si potrebbe incontrare è quello di avere dei contenuti composti da codice HTML. In quel caso possiamo usare una sezione `CDATA` per istruire il parser XML a trattare il contenuto dell'elemento come testo semplice, ignorando quindi gli eventuali tag HTML presenti al suo interno.