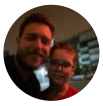
Node.js Express (4.x) EJS — Reaproveitando Views Utilizando a Função include do EJS

Todo desenvolvedor sabe que duplicar código é uma péssima ideia. Já parou para pensar que isto também pode ser um problema na estrutura de suas views?



Marcelo Vismari
Nov 24, 2019 · 4 min read

Photo by Caspar Camille Rubin on Unsplash

Organizar seu projeto envolve muitos cuidados e atenção. Ao longo do desenvolvimento sempre nos deparamos em situações onde é possível melhorar algumas coisas.

Se tratando de um projeto na arquitetura MVC, como o Express, é natural que a quantidade de Views cresça conforme o projeto evolui e é neste ponto que podemos evitar a repetição de código criando “pedaços” de Views que possam ser utilizadas para compor outras Views. Confuso?

Suponha que estejamos trabalhando no desenvolvimento de um ecommerce. Uma das tarefas mais comuns ao se codificar um sistema deste tipo é listar os produtos que o usuário comprou ou que está em seu carrinho. Esta lista pode aparecer em diversos locais como por exemplo:

- na tela do carrinho no inicio do processo de checkout
- na conclusão da tela de checkout indicando quais itens foram comprados
- na consulta do histórico de compras
- no email que é enviado ao cliente com os produtos que ele comprou

Veja que uma simples **lista de produtos** aparece em várias partes do sistema.

Cenário

Para deixar o cenário mais realista, suponha que na View de checkout tenhamos algo do tipo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- ... -->
    <title>Node.js - Express - EJS</title>
</head>
<body>
    <p>Olá Sr(a). <%= usuario.nome %> ...</p>

    <!-- ... mais informações da página ... -->

    <table>
        <thead>
            <tr>
                <th>Item</th>
                <th>Quantidade</th>
                <th>Valor</th>
            </tr>
        </thead>
        <tbody>
        <% for(let item of produtos) { %>
            <tr>
                <td><%= item.nome %></td>
                <td><%= item.quantidade %></td>
                <td><%= item.valor %></td>
            </tr>
        <% } %>
        </tbody>
        <tfoot>
```

```
                <tr>
                    <td></td>
                    <td></td>
                    <td>
                        <%=
                            produtos
                                .map(a => a.valor)
                                .reduce((acc, cur) => acc + cur);
                        %>
                    </td>
                </tr>
            </tfoot>
        </table>

        <!-- ... mais informações da página ... -->
    </body>
    </html>
```

> *Obs.: a função `map` irá retornar um array com o valor de cada produto do array `produtos`. Na sequência utilizamos a função `reduce` para somar o valor dos produtos.*

E no controller algo do tipo:

```
(req, res) => {
  const model = {
    usuario: {
      nome: 'Teste 123'
    },
    produtos: [{
      nome: 'Produto 1',
      quantidade: 2,
      valor: 30.00
    },
    {
      nome: 'Produto 2',
      quantidade: 1,
      valor: 40.00
    },
    {
      nome: 'Produto 3',
      quantidade: 3,
      valor: 9.90
    }
  ]};

  res.render('checkout/index', model);
}
```

Olá Sr(a) Teste 123

Olá Sr(a). Teste 123 ...

Item	Quantidade	Valor
Produto 1	2	30
Produto 2	1	40
Produto 3	3	9.9
		79.9

Resultado da renderização do código acima

Problema

Repare que o trecho que renderiza os produtos é o trecho que está dentro do `<table>...</table>`. É exatamente este trecho que será repetido em outras views para apresentar uma **lista de produtos**. Se repetirmos este trecho em outras views teremos alguns problemas, como por exemplo:

- A manutenção ficará mais custosa visto que o desenvolvedor terá que alterar várias Views onde há uma **lista de produtos**. Exemplo: suponha que haverá uma alteração na estrutura da `table`. Esta alteração deverá ser feita em toda View que tiver este trecho de código;
- Aumento desnecessário na quantidade de linhas das Views já que o código é copiado e colado em cada View que quiser renderizar uma **lista de produtos**;

Solução (função include)

Então o primeiro passo é separar a responsabilidade das Views garantindo que cada View terá uma única responsabilidade de renderizar um determinado tipo de modelo. No nosso exemplo, vamos criar uma View que irá renderizar uma lista de produtos e nada mais. Então podemos codificar nossa View da seguinte forma:

```
<table>
    <thead>
        <tr>
            <th>Item</th>
            <th>Quantidade</th>
            <th>Valor</th>
```

```
            </tr>
        </thead>
        <tbody>
        <% for(let item of produtos) { %>
            <tr>
                <td><%= item.nome %></td>
                <td><%= item.quantidade %></td>
                <td><%= item.valor %></td>
            </tr>
        <% } %>
        </tbody>
        <tfoot>
            <tr>
                <td></td>
                <td></td>
                <td>
                    <%=
                        produtos
                            .map(a => a.valor)
                            .reduce((acc, cur) => acc + cur);
                    %>
                </td>
            </tr>
        </tfoot>
    </table>
```

Veja que este é o único código da view (partial-product-detail.ejs). Ele espera que um Array `produtos` seja passado como modelo para renderizar a **lista de produtos**.

Isolando o código que renderiza uma lista de produtos em uma view parcial (parcial porque ela só renderiza um pedaço do HTML que compõem uma página) podemos incluí-la dentro de uma outra view através da utilização a função `include(caminho_da_view, dados)`:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- ... -->
    <title>Node.js - Express - EJS</title>
</head>
<body>
    <p>Olá Sr(a). <%= usuario.nome %> ...</p>

    <!-- ... mais informações da página ... -->

    <%- include('../product/partial-product-list', { produtos }); %>

    <!-- ... mais informações da página ... -->
</body>
</html>
```

Veja que a view de checkout citada mais acima ficou bem menor e muito mais legível. Com esta solução qualquer outra View que precise renderizar uma **lista de produtos** poderá realizar esta tarefa através da inclusão de apenas uma linha:

```
<%- include('../product/partial-product-list', { produtos }); %>
```

> *Obs.:* `{ produtos }` *é a mesma coisa que* `{ produtos: produtos }`.

Isso irá facilitar e muito a manutenção pois a responsabilidade de renderizar os produtos, agora, está em um único lugar.

Conclusão

Agora imagine organizar todas as suas views desta forma, garantindo que cada uma terá uma única responsabilidade. Podemos separar uma View parcial para renderizar cada pedaço do seu sistema como rodapé, cabeçalho, listas, detalhes e no final a legibilidade/manutenção ficará muito melhor. Exemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <%- include('../shared/css/styles') %>
    <title>Node.js - Express - EJS</title>
</head>
<body>
    <div>
        <!-- Dados do usuário -->
        <%- include('../user/user-detail', usuario) %>
    </div>

    <!-- Lista de produtos -->
    <%- include('../product/partial-product-list', produtos) %>

    <footer>
        <!-- Renderiza o rodapé -->
        <%- include('../common/footer') %>
    </footer>

    <!-- Inclui alguns scripts -->
    <%- include('../shared/js/scripts') %>
</body>
</html>
```

Sugestão de leitura: documentação do EJS: https://ejs.co/#docs

Ejs Nodejs Expressjs Mvc Frameworks Web Development

About Help Legal

Get the Medium app