07

Organização do projeto

Transcrição

O objetivo agora é deixarmos nosso projeto estruturado corretamente, de forma que possamos evoluir com ele durante o curso. Além disso, aprenderemos como contornar o "problema" que vimos anteriormente: nosso sistema retorna a mesma página para toda e qualquer requisição feita a partir de http://localhost:3000 (http://localhost:3000).

Queremos possibilitar que a aplicação devolva respostas diferentes para cada requisição feita pelo usuário no cliente. Primeiramente, precisaremos verificar qual é o pedido que está sendo feito pelo usuário. Para isso, faremos uma verificação a partir da requisição, buscando a URL solicitada.

Vamos declarar, dentro da função *callback*, uma variável `html` que inicialmente não receberá valor algum.

```
let html = '';
```

Então, se essa URL for igual à raiz da nossa aplicação (`if (req.url == '/' )`, devolveremos o trecho de HTML que criamos anteriormente. Ao final, a resposta deverá se basear na variável `html` :

```
const http = require('http');

const servidor  = http.createServer(function (req, resp) {

    let html = '';
    if (req.url == '/') {
        html = `
        <html>
            <head>
                <meta charset="utf-8">
            </head>
            <body>
                <h1> Casa do Código </h1>
            </body>
        </html>
    `;
    }
    resp.end(html);
});
servidor.listen(3000);
```

Também queremos ser capazes de atender outros caminhos na nossa aplicação. Em uma aplicação web, esses caminhos são chamados de **"rotas"**. Podemos criar uma rota acrescentando um `else if (req.url == '/livros')` - ou seja, buscando a propriedade `/livros` na URL da requisição. Em seguida, atribuiríamos um novo valor à variável `html` :

```
const servidor  = http.createServer(function (req, resp) {

    let html = '';
    if (req.url == '/') {
```

```
        html = `
        <html>
            <head>
                <meta charset="utf-8">
            </head>
            <body>
                <h1> Casa do Código </h1>
            </body>
        </html>
    `;
    } else if (req.url == '/livros')
        html = `
        <html>
            <head>
                <meta charset="utf-8">
            </head>
            <body>
                <h1> Listagem de livros </h1>
            </body>
        </html>
    `;

    resp.end(html);
```

Dessa forma, se quiséssemos outras rotas, precisaríamos adicionar outros `else if ()` sucessivamente. Por enquanto, faremos dessa maneira, mas talvez não seja a melhor abordagem para nossa aplicação.

No Prompt de Comando, executaremos novamente o servidor. Acessando a URL http://localhost:3000/livros (http://localhost:3000/livros), teremos a resposta:

> Listagem de livros

Exatamente como esperávamos, certo? Porém, temos um problema de **complexidade ciclomática**: quanto mais caminhos o programa tiver, mais complexo e de difícil manutenção o seu código será.

Teremos, então, que reduzir ao máximo a quantidade de `if` na nossa aplicação, o que torna a nossa abordagem atual bastante inadequada.

Para ilustrar ainda melhor esse problema, considere que, quando passamos uma URL no navegador, por exemplo http://localhost:3000/livros (http://localhost:3000/livros), ele faz uma requisição do tipo GET. Existem outros métodos além desse, como POST, PUT e DELETE.

Até o momento, nosso servidor só está respondendo requisições do tipo GET. Se precisássemos criar condições para cada tipo de requisição, nosso código se tornaria ainda mais complexo e trabalhoso.

Pensando nesses problemas relacionados ao desenvolvimento de aplicações web que foram surgindo os módulos no ambiente Node, dentre eles o famoso `express`, um *framework* que nos trará algumas facilidades.

Antes de implementarmos esse *framework* na nossa aplicação, temos que transformá-la em uma aplicação do tipo Node, já que, atualmente, só temos uma aplicação JavaScript que é executada com o Node.

Para isso, executaremos no Prompt o comando `npm init` (o inicializador do *node package manager*, que é o gerenciador de pacotes do Node). Pressionando "Enter", será exibida uma sequência de perguntas na tela. Manteremos

o nome do pacote como `casadocodigo` e a versão como `1.0.0`. Como descrição, escreveremos "Livraria Casa do Codigo".

Já no ponto de entrada (*entry point*), alteraremos para `server.js`. Como não temos um comando de teste (*test command*) ou um repositório no Git (*git repository*), deixaremos esses campos em branco. As palavras chave (*keywords*) também podem ser deixadas em branco.

No nome do autor, você pode preencher o seu próprio nome. Nesse caso, preenchemos como `Aluno`. A licença deverá ser mantida como padrão (`ISC`).

Ao final, o inicializador fará uma nova verificação. Pressionando "Enter", ele será executado, criando um arquivo `package.json` na pasta do nosso projeto, com todas as informações que acabamos de preencher. Vamos analisá-lo:

```
{
  "name": "casadocodigo",
  "version": "1.0.0",
  "description": "Livraria Casa do Codigo",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "author": "Aluno",
  "license": "ISC"
}
```

Esse arquivo é muito importante, pois define o nosso projeto como sendo do tipo Node, e finalmente poderemos instalar o `express`. No Prompt, digitaremos o comando `npm install express@4.16.3` - ou seja, instalaremos o `express` com o gerenciador de pacotes, mas especificamente a versão `4.16.3`. Dessa forma, manteremos a sincronia entre o conteúdo apresentado no curso e a sua prática.

Além disso, adicionaremos a especificação `--save-exact` para expressarmos que esse pacote (e essa versão em específico) é uma dependência da nossa aplicação (ou seja, que ela obrigatoriamente necessita dele para funcionar).

Com isso, teremos:

```
npm install express@4.16.3 --save-exact
```

Após pressionar o "Enter", o pacote será baixado e instalado no diretório em questão. Terminado o download, uma nova seção terá sido adicionada no nosso `package.json`:

```
{
  "name": "casadocodigo",
  "version": "1.0.0",
  "description": "Livraria Casa do Codigo",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "author": "Aluno",
```

```
  "license": "ISC",
  "dependencies": {
    "express": "4.16.3"
  }
}
```

A seção `dependencies` lista justamente as dependências do nosso projeto - nesse caso, somente o `express` na versão que definimos. Por fim, criaremos uma pasta chamada "src" (de "*source*") na raiz do nosso projeto, na qual armazenaremos todo e qualquer código relativo a nossa aplicação.

Dentro dela, criaremos outras duas pastas:

- "app", na qual armazenaremos os códigos relativos à lógica
- "config", na qual serão armazenados códigos de configuração

Agora temos o ambiente pronto e estabelecido para prosseguirmos com os estudos. Vamos lá?