02

Express em ação

Transcrição

O objetivo agora é fazermos o pacote `express` funcionar na nossa aplicação. Primeiramente, não queremos mais lidar diretamente com o protocolo HTTP, já que essa função será delegada ao `express`.

Portanto, no início do arquivo `server.js`, criaremos uma constante `express` que receberá a chamada para o módulo de mesmo nome (`require('express')`). O retorno desse método será uma função.

Chamaremos essa função com `express()`, obtendo um objeto do tipo `express` que poderemos utilizar para configurar a nossa aplicação.

```
const app = express();
```

De posse da constante `app`, estaremos habilitados a criar nosso servidor utilizando o `express`. Para isso, executaremos o método `listen()`, que recebe como primeiro parâmetro o número da porta na qual o servidor rodará (`3000`). Também poderíamos indicar um `hostname`, o que não queremos no momento, e receber uma função *callback*.

Quando criamos o servidor usando o protocolo HTTP, definimos um *callback* que seria executado sempre que recebêssemos uma requisição web. No caso do `express`, esse *callback* só será executado quando nosso servidor for iniciado.

Nesse momento, escreveremos a mensagem "Servidor rodando na porta 3000" na tela.

```
app.listen(3000, function() {
    console.log('Servidor rodando na porta 3000');

})
```

Feitas essas configurações, comentaremos todo o código que escrevemos anteriormente. No terminal, executaremos o comando `node server.js`. Ao pressionarmos "Enter", a mensagem que preparamos será exibida.

Se tentarmos acessar a URL http://localhost:3000 (http://localhost:3000), receberemos a mensagem "Cannot GET /". Isso significa que o `express` não está habilitado a devolver uma resposta do método GET no endereço `/` da nossa aplicação, afinal somente criamos o servidor, sem definir nenhuma rota.

Voltando ao `server.js`, pegaremos a constante `app` e executaremos o método `get()` - já que, obviamente, estamos tentando atender a requisições desse tipo. Esse método receberá dois parâmetros: uma *string* representando o caminho que queremos atender (nesse caso, `/`, que é a rota raiz), e uma função do tipo *callback* que será executada sempre que o cliente acessar o endereço `/`.

Essa função receberá a requisição e a resposta (`req, resp`), e seu corpo indicará a resposta enviada ao usuário. Da mesma forma que fizemos quando estávamos trabalhando com o módulo HTTP, passaremos o objeto representando a resposta (`resp`) e executaremos o método `send()`, recebendo como parâmetro uma *string* que representa o HTML.

```
app.get('/', function(req, resp) {
    resp.send(
        `
            <html>
                <head>
                    <meta charset="utf-8">
                </head>
                <body>
                    <h1> Casa do Código </h1>
                </body>
            </html>
        `
    );
});
```

Executando novamente nosso servidor, o texto "Casa do Código" passará a ser exibido em http://localhost:3000 (http://localhost:3000).

Para criarmos uma nova rota na nossa aplicação, basta repetirmos o processo. Por exemplo, vamos criar uma rota `/livros` com o texto "Listagem de livros":

```
app.get('/livros', function(req, resp) {
    resp.send(
        `
            <html>
                <head>
                    <meta charset="utf-8">
                </head>
                <body>
                    <h1> Listagem de livros </h1>
                </body>
            </html>
        `
    );
});
```

Reiniciando a aplicação e acessando a URL http://localhost:3000/livros (http://localhost:3000/livros), a mensagem que definimos será exibida na tela. Porém, ainda precisamos nos atentar a alguns detalhes importantes sobre o nosso código.

Primeiramente, de onde vem o módulo `express` ? Anteriormente, nós o instalamos, fazendo com que ele aparecesse na lista de dependências de `package.json` .

Quando instalamos o `express` , o Node criou uma pasta chamada "node_modules" dentro da nossa aplicação. Essa pasta é muito importante, pois abrigará todas as dependências da nossa aplicação. Além disso, as dependências baixadas serão salvas de acordo com o sistema operacional que estivermos utilizando.

Dessa forma, se passarmos uma pasta "node_modules" do Windows para uma máquina com Mac ou Linux, a aplicação não funcionará. Por isso, antes de compartilhamos projetos Node, é uma boa prática remover essa pasta. Como o arquivo `package.json` tem a listagem de dependências, bastará pedir para que o Node faça o download delas novamente.

Para isso, basta executar o comando `npm install` no terminal do sistema operacional. Esse comando irá procurar um arquivo `package.json` dentro da pasta, lendo-o e baixando todas as dependências do projeto, criando novamente a pasta "node_modules".

Sabendo disso, sempre que você fizer o download dos arquivos do curso, também será necessário executar esse comando!

No nosso `server.js`, quando executamos o método `require('express')`, o Node procura um módulo com esse nome na pasta "node_modules". Se o módulo não for encontrado, teremos um erro.

Ainda temos um problema: o código que escrevemos está funcionando como esperávamos, mas ele ainda não tem um *design* correto, que possibilite evoluirmos a aplicação de maneira sustentável. Aprenderemos a melhorar nosso código a seguir!