05

Encapsulando as rotas

Transcrição

Dando continuidade às refatorações do nosso código, o objetivo agora é encapsular as rotas da aplicação em um arquivo específico.

Dentro da pasta "app" (que está dentro da pasta "src"), criaremos uma nova pasta "rotas", onde ficarão todos os arquivos referentes a rotas da aplicação. Nesta pasta, criaremos o arquivo `rotas.js`, que abrigará o código relativo aos caminhos que escrevemos anteriormente.

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

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

Ainda precisamos declarar a importação desse módulo de rotas. Porém, não faremos isso no arquivo `server.js`, pois queremos manter nele somente a criação do servidor.

Ou seja, essa importação será feita no `custom-express.js`, onde é feita a customização do módulo. Após a criação da constante `app`, vamos declarar `require()`, pedindo a importação do arquivo de rotas:

```
require('../app/rotas/rotas.js');
```

Se tentarmos executar o `server.js` nesse ponto, receberemos um erro, já que o módulo `rotas.js` possui um objeto `app` que em nenhum momento foi declarado.

No `custom-express.js`, criaremos uma constante `rotas` que será retornada pela importação desse módulo. A partir dela, passaremos o objeto `app` como parâmetro.

```
const rotas = require('../app/rotas/rotas.js');
rotas(app);
```

No arquivo `rotas.js`, precisaremos exportar uma função (que, no mundo JavaScript, é a instrução capaz de receber um parâmetro) capaz de receber o objeto `app`. É possível fazer isso com `module.exports`.

Criaremos essa função utilizando a sintaxe do **ECMAScript 6**, que introduziu as famosas *arrow functions*:

```
module.exports = (app) => {
}
```

Para nosso código ficar mais elegante, passaremos a construção das rotas para dentro da declaração do `module.exports`:

```
module.exports = (app) => {

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

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

}
```

Agora podemos salvar todas as alterações na nossa aplicação e executar o `server.js`. A aplicação funcionará corretamente, mostrando os textos que definimos anteriormente em suas respectivas URLs.

Dessa maneira, não só conseguimos separar as responsabilidades do nosso código em diferentes módulos (o que é uma boa prática), como também aprendemos que a sintaxe do `module.exports` não só permite a exportação de objetos, como também de funções que recebam parâmetros.

Durante o curso continuaremos melhorando nosso código, mas nosso próximo objetivo é melhorar o processo de codificação e teste da nossa aplicação.