02

Acessando o Banco de dados

Transcrição

Vamos dar continuidade ao nosso projeto de modo que finalmente consigamos tornar a nossa listagem de livros dinâmica, ou seja, fazendo acesso a um banco de dados.

Antes de tudo, no terminal, executaremos o comando `npm install sqlite3@4.0.2 --save-exact` para instalar o banco de dados que utilizaremos no curso. Com o SQLite habilitado, precisaremos colocar o arquivo `database.js` (https://caelum-online-public.s3.amazonaws.com/980-nodejs-fundamentos/03/database.js) dentro da pasta "config" do nosso projeto.

Esse arquivo importará o SQLite e criará uma instância do nosso banco de dados:

```
const sqlite3 = require('sqlite3').verbose();
const bd = new sqlite3.Database('data.db');
```

Com essa sintaxe, o Node criará um arquivo `data.db` dentro da pasta do nosso projeto, representando o banco de dados que iremos utilizar.

Ao final, ele disponibilizará a instância do banco de dados por meio do módulo que o nosso arquivo `database.js` representa, de modo que possamos manipular esse banco em outros arquivos.

> Nota: Por enquanto você não precisa se preocupar com a motivação do uso do SQLite, nem com detalhes do funcionamento do `database.js` .

No arquivo `rotas.js` , vamos criar uma constante `db` recebendo `require()` , passando o caminho do módulo onde o banco de dados foi configurado:

```
const db = require('../../config/database')
```

Dessa forma, teremos uma instância desse banco de dados. Em seguida, na rota `/livros` , escreveremos `db.all()` , que é um método do SQLite executado quando queremos fazer algum tipo de listagem. Esse método receberá dois parâmetros: o primeiro, uma *string* representando a consulta que queremos fazer; e o segundo, uma função *callback* que será executada quando a nossa consulta tiver terminado.

Essa função receberá dois parâmetros: `erro` , se ocorrer algum erro nessa consulta; e `resultados` . Como queremos selecionar todos os dados da tabela `livros` , a *string* que passaremos será `SELECT * FROM livros` :

```
app.get('/livros', function(req, resp) {
    db.all('SELECT * FROM livros', function(erro, resultados) {

    });
```

Se a nossa consulta der certo, os livros retornados serão armazenados no parâmetro `resultados` . Então, continuaremos chamando o *template* de listagem e passando os livros para a nossa página. Porém, dessa vez, os itens dessa lista não

serão estáticos, mas sim os `resultados` vindos do banco de dados.

Desse modo, passaremos o código do método `marko()` para dentro da função *callback*, substituindo os itens estáticos da lista pelos `resultados` :

```
app.get('/livros', function(req, resp) {
    db.all('SELECT * FROM livros', function(erro, resultados) {

        resp.marko(
            require('../views/livros/lista/lista.marko'),
            {
                livros: resultados
            }

        );

    });


});
```

Feito isso, podemos salvar a aplicação e executá-la novamente no terminal. Acessando http://localhost:3000/livros (http://localhost:3000/livros), encontraremos a mesma lista que criamos anteriormente, mas dessa vez sendo acessada diretamente do banco de dados:

Listagem de livros

ID Título

1 Node na prática

2 JavaScript na prática

Precisamos nos atentar ao fato de que o arquivo `rotas.js` está acumulando muitas possibilidades: estamos chamando a instância do banco de dados, selecionando os dados desse banco, acessando o *template* e finalmente passando oa lista para a página no navegador.

Já sabemos que isso não é uma boa prática, mas como melhorar nosso código, distribuindo essas responsabilidades? A solução virá a seguir!