05

DAO com Promises

Transcrição

Agora continuaremos melhorando o DAO que criamos anteriormente. Primeiro, vamos analisar o código da rota `/livros` em `rotas.js`:

```
app.get('/livros', function(req, resp) {

    const livroDao = new LivroDao(db);

    livroDao.lista(function(erro, resultados) {

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

        );

    });

});
```

Quando fazemos a listagem, chamamos o método `lista()` do nosso `livroDao`, passando uma função de *callback* que é executada ao final da operação assíncrona do acesso ao banco de dados. Por padrão, no mundo JavaScript, utilizam-se *Promises* para esse tipo de situação, e é isso que faremos agora.

Começaremos chamando o método `livroDao.lista()` e, em seguida, o método `then()` das *Promises* (já conhecido por quem fez os cursos de JavaScript aqui na plataforma).

Esse método deve receber a lista de `livros` e, a partir disso, enviá-la para a página de listagem com o método `resp.marko()` que construímos anteriormente. Porém, ao invés de `resultados`, passaremos `livros`, que é o parâmetro que estamos recebendo do método `then()`.

Além disso, também podemos concatenar a chamada do método `catch()`, que é executado quando acontece algum erro no processamento de uma *Promise*. Passaremos então o `erro` e pediremos que ele seja impresso com `console.log(erro)`.

Assim, teremos:

```
app.get('/livros', function(req, resp) {

    const livroDao = new LivroDao(db);
    livroDao.lista()
            .then(livros => resp.marko(
                require('../views/livros/lista/lista.marko'),
                {
                    livros: livros
```

```
                }

            ))
            .catch(erro => console.log(erro));

});
```

Agora, em `livro-dao.js`, precisaremos alterar o método de listagem, que não mais receberá uma função *callback*. No corpo dele, criaremos uma nova *Promise* com `return new Promise()`.

Com a sintaxe das *arrow functions*, faremos com que essa *Promise* receba uma função de dois parâmetros: `resolve` e `reject`, ou seja, as funções que serão executadas no momento que **resolvermos** ou **rejeitarmos** nossa *Promise*.

Além disso, o corpo da *Promise* também será composto pela operação assíncrona que queremos fazer: o acesso ao banco de dados para fazer a listagem, mas sem a função `callback()`.

No lugar dela, retornaremos um erro com `if (erro) return reject('Não foi possível listar os livros!')`. O `return` é para que o código pare de ser executado caso a operação seja rejeitada.

Caso isso não tenha acontecido, finalizaremos a *Promise* com sucesso, passando os `resultados` (que armazena a listagem de livros) como parâmetro de `resolve()`. Como nossa *arrow function* tem mais de um comando, precisamos definir o corpo dela com chaves (`{}`). Assim, teremos:

```
class LivroDao {

    constructor(db) {
        this._db = db;
    }

    lista() {
        return new Promise((resolve, reject) => {
            this._db.all(
                'SELECT * FROM livros',
                (erro, resultados) => {
                    if (erro) return reject('Não foi possível listar os livros!');

                    return resolve(resultados);
                }
            )

        });
    }
}

module.exports = LivroDao;
```

Feito isso, podemos rodar novamente nossa aplicação. Na URL http://localhost:3000/livros (http://localhost:3000/livros), continuaremos recebendo a nossa lista:

Listagem de livros

ID Título

1 Node na prática

2 JavaScript na prática

Finalizado esse processo, já estamos aptos a desenvolver a página de cadastro de livros, de modo a aumentar a nossa listagem. É isso que aprendemos a seguir!