02

# Remoção de livros

### Transcrição

Daremos continuidade ao desenvolvimento da nossa aplicação, implementando a funcionalidade de remoção de livros da nossa listagem. De volta ao nosso código, alteraremos mais três colunas no HTML da listagem de livros: o "Preço", um link para "Editar" e um link para "Remover".

```
<html>
    <head>
        <meta charset="utf-8">
    </head>
    <body>
        <h1> Listagem de livros </h1>

        <table id="livros">
            <tr>
                <td>ID</td>
                <td>Título</td>
                <td>Preço</td>
                <td>Editar</td>
                <td>Remover</td>
            </tr>
            <tr id="livro_${livro.id}" for (livro in data.livros)>
                <td>${livro.id}</td>
                <td>${livro.titulo}</td>
                <td>${livro.preco}</td>
                <td><a href="#">Editar</a></td>
                <td><a href="#" data-ref="${livro.id}" data-type="remocao">Remover</a></td>
            </tr>
        </table>

        <script src="./remove-livro.js">
        </script>
    </body>
</html>
```

Em cada uma das `<tr>`, colocamos uma `id` com a informação `livro_` seguida do respectivo `id` desse livro. Além disso, no link de remoção, utilizamos a *dataset* API do HTML para informar que esse link faz referência a um determinado livro, passando o `id` dele.

Perceba que a propriedade `href` do link de remoção recebeu apenas um `#`, e não uma URL indicando para onde será feita a requisição. Isso porque é o JavaScript que terá a responsabilidade de enviar a requisição para nosso servidor, que efetivamente removerá o livro do banco de dados e devolverá uma resposta dizendo se a remoção ocorreu com sucesso ou não.

De posse dessa resposta, o JavaScript atualizará (ou não) a listagem, evitando que seja necessário recarregar toda a listagem novamente. Justamente por isso, temos declarado um arquivo de JavaScript chamado `remove-livro.js` (localizado na mesma pasta do arquivo de listagem), responsável por enviar a requisição para o servidor e remover o livro.

Nesse arquivo, selecionaremos a `tabelaLivros` e adicionaremos nela um ouvinte para o evento de clique. Se o elemento que foi clicado tiver o *data type* `remocao`, saberemos que é o nosso link de remoção. Em seguida, pegaremos o `dataref` com o `Id` do nosso livro, e com a `fetch()` do JavaScript, faremos uma requisição para a URL `/livros/${livroID}` (que será um valor numérico).

Além disso, nossa requisição terá que ser feita com o método `DELETE` do HTTP. Se tivermos uma resposta positiva do servidor, removeremos a linha da tabela referente a aquele ID. Se houver um problema, será feito um `console.log()` do erro.

```
let tabelaLivros = document.querySelector('#livros');
tabelaLivros.addEventListener('click', (evento) => {
    let elementoClicado = evento.target;

    if (elementoClicado.dataset.type == 'remocao') {
        let livroId = elementoClicado.dataset.ref;
        fetch(`http://localhost:3000/livros/${livroId}`, { method: 'DELETE' })
            .then(resposta => {

                let tr = elementoClicado.closest(`#livro_${livroId}`);
                tr.remove();

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

    }

});
```

Todo esse processo se dará do lado do navegador. Como serão feitas requisições para `http://localhost:300/livros/${livroId}`, precisaremos criar uma rota que consiga atender à essa requisição para o método DELETE. No arquivo `rotas.js`, escreveremos o método `app.delete()`, seguido da URl que queremos acessar e da função *callback* que recebe `req` e `resp`.

Precisamos que o `express` saiba que a informação que estamos passando na URL é a `id` do livro que queremos excluir, e que essa informação é variável para cada requisição. Felizmente, o `express` nos possibilita criar variáveis na própria URL da rota. Para isso, basta usarmos dois pontos ( `:` ) seguidos do nome da variável, que nesse caso é `id`.

Para recuperarmos esse valor, basta, dentro do *callback*, buscarmos o `id` entre os parâmetros dentro da requisição ( `req.params.id` ). Essa informação será salva em uma constante `id`.

```
app.delete('/livros/:id', function(req, resp) {
    const id = req.params.id;
});
```

Poderemos, então, criar uma nova constante `livroDao`, responsável por fazer o acesso ao banco de dados. Em seguida, chamaremos o método de remoção do nosso `livroDao`, recebendo o `id`:

```
app.delete('/livros/:id', function(req, resp) {
    const id = req.params.id;

    const livroDao = new LivroDao(db);
```

```
    livroDao.remove(id)
});
```

Dessa vez, se tudo der certo, não queremos mostrar uma nova página ao usuário, mas apenas informá-lo que a requisição de remoção funcionou - ou seja, devolver um *status* HTTP de valor `200`.

Faremos isso com o método `resp.status(200)`, finalizando essa resposta com o já conhecido `end()`. Caso algo dê errado, usaremos `catch(erro => console.log(erro))`. Assim, teremos:

```
app.delete('/livros/:id', function(req, resp) {
    const id = req.params.id;

    const livroDao = new LivroDao(db);
    livroDao.remove(id)
        .then(() => resp.status(200).end())
        .catch(erro => console.log(erro));

});
```

Feito isso, salvaremos as alterações e iniciaremos novamente a aplicação. Acessando a URL http://localhost:3000/livros (http://localhost:3000/livros), clicaremos no botão "Remover" ao lado de qualquer um dos livros na lista. Porém, a única resposta que teremos será a mudança para a URL http://localhost:3000/livros# (http://localhost:3000/livros#).

No console do navegador ("F12"), receberemos alguns erros:

> Failed to load resource: the server responded with a status of 404 (Not Found)
>
> Refused to execute script from 'http://localhost:3000/remove-livro.js' (http://localhost:3000/remove-livro.js') because its MIME type ('text/html') is not executable, and strict MIME type checking is enabled

Entenderemos esses erros a seguir!