02

Usando AJAX

Nosso próximo passo agora é marcar uma conta como "paga". Mas dessa vez faremos diferente: ao invés de fazermos isso pelo formulário de alteração, criaremos um link na própria listagem, com o texto "Pagar agora!" que, ao ser clicado, marcará a conta como paga.

Fazer isso não tem segredo algum. Basta fazer esse link apontar para uma Action no controller, que invocará o método `paga()` do DAO, e redirecionar novamente para a listagem. Só que ao fazer isso, temos a sensação da tela "piscar". Claro, uma requisição aconteceu.

Nosso objetivo aqui é fazer com que essa requisição aconteça por "baixo dos panos". Ou seja, vamos clicar no link, a requisição vai acontecer, ela vai voltar para o navegador, mas a tela não vai piscar. Sim, isso é possível, e o nome disso é **AJAX**.

Ajax é a tecnologia que nos possibilita fazer requisições assíncronas, e tratar suas respostas. AJAX significa `Assynchronous Javascript and XML`. Apesar de ter XML no nome, não precisamos usá-lo. Mas sim, precisaremos usar Javascript para isso. Para facilitar nossa vida, também utilizaremos a biblioteca JQuery, que nos ajudará e muito na hora de fazer requisições Ajax.

Com JQuery, fica fácil fazer requisições Ajax. Por exemplo, se quiséssemos fazer uma requisição para a página "/formulario", bastaria fazermos:

```
$.get("/formulario");
```

O método `$.get()` esconde toda a complexidade dessa requisição. E é isso que vamos fazer: vamos fazer uma requisição para uma action de um de nossos controllers, passando o ID da conta a ser marcada como paga. Vamos começar por essa função Javascript:

```
function pagaAgora(id) {
  $.get("pagaConta?id="+id);
}
```

Veja que só com as linhas de código acima, já fizemos uma requisição para a URL `pagaConta?id=NUMERO QUALQUER`. Mas temos um problema: a requisição acontecerá, e quando a resposta voltar, nada acontecerá na tela. Precisamos ao menos exibir uma mensagem para o usuário, pra ele saber que a operação deu certo.

A função `$.get()` permite que passemos um parâmetro a mais pra ela. Esse parâmetro é uma função, que será executada quando a requisição voltar. Veja o código abaixo. Quando a requisição voltar, a função `deuCerto()` será executada.

```
function deuCerto(dadosDaResposta) {
  alert("Conta paga com sucesso!");
}
function pagaAgora(id) {
  $.get("pagaConta?id="+id, deuCerto);
}
```

Precisamos agora invocar essa função Javascript de algum lugar. Vamos criar um link, parecido com aquele de Remover, que tínhamos na tabela. Só que agora, repare no "onclick". Ele vai invocar a função javascript:

```
<td>
    <a href="removeConta?id=${conta.id}">Deletar</a> |
    <a href="#" onclick="pagaAgora(${conta.id});">Pagar</a>
</td>
```

Veja que ali passamos `pagaAgora(${conta.id})`, ou seja, o próprio link deve ser gerado dinamicamente, pois ele é diferente para cada linha da tabela.

Vamos agora para nosso Controller. A action é bem parecida com as que já escrevemos até então:

```
@RequestMapping("/pagaConta")
public String paga(Long id) {
  ContaDAO dao = new ContaDAO();
  dao.paga(id);

  return "uma-pagina-qualquer";
}
```

Mas um problema que temos agora é: o que devolver? Pra qual JSP redirecionar? Afinal, lembre-se que essa requisição está acontecendo por baixo dos panos, e o que for devolvido pelo Spring MVC não será exibido no browser!

O que podemos fazer é então devolver apenas um "SUCESSO". Algo que o browser saiba que a requisição funcionou. No HTTP, temos uma maneira elegante de fazer isso. Basta dizer que o `status code` da requisição é 200. Você já viu outros status code por aí: quando você acessa uma página que não existe, a mensagem que aparece é "404". Quando você vê uma aplicação que não funciona, o retorno é "500". E assim por diante.

Para fazer isso no Spring MVC, precisamos receber um `HttpServletResponse` na Action, e dizer que o status dele é 200:

```
@RequestMapping("/pagaConta")
public void paga(Long id, HttpServletResponse response) {
  ContaDAO dao = new ContaDAO();
  dao.paga(id);
  response.setStatus(200);
}
```

Excelente. Para que tudo isso funcione, precisamos incluir as bibliotecas do JQuery em nosso projeto. Para isso, basta acrescentarmos a linha abaixo no começo do nosso JSP:

```
<script src="resources/js/jquery.js"></script>
```

Lembre-se que o JQuery é uma biblioteca, e você pode baixá-la na internet, através do site oficial (www.jquery.com). Por fim, precisamos mexer novamente no `spring-context.xml`. O Spring está preparado para responder toda e qualquer requisição que chegar pra ele. Mas precisamos dizer que em algumas delas ele não tem o que fazer, como é o caso da requisição que o browser fará para pegar o arquivo `jquery.js`. Para isso, basta colocarmos todos esses arquivos que devem apenas serem "servidos" dentro do diretório `WebContent/resources/`, e incluirmos a seguinte linha no arquivo de configuração do Spring MVC:

```
<mvc:default-servlet-handler/>
```

Excelente. Agora nossa aplicação faz requisições usando Ajax. Veja a quantidade de possibilidades que isso nos traz! Veja como podemos fazer interfaces extremamente amigáveis para nossos usuários finais com apenas algumas linhas de código. Acostume-se com Ajax e JQuery, você os utilizará bastante!