03

Salvando notas do aluno

Transcrição

Após aprendermos como cadastrar as habilidades dos alunos, faremos o mesmo em relação a suas notas. Como habilidades, notas também possui domínio diferente e, por isso, merece seu próprio *controller*. Semelhante ao de habilidades, teremos:

```
@Controller
public class NotaController {

  @Autowired
  private AlunoRepository repository;

  @GetMapping("/nota/cadastrar/{id}")
  public String cadastrar(@PathVariable String id, Model model) {
    Aluno aluno = repository.obterAlunoPor(id);
    model.addAttribute("aluno", aluno);
    model.addAttribute("nota", new Nota());

    return "/nota/cadastrar";
  }

}
```

Precisaremos tanto do objeto `aluno` quanto do de nota. Dessa forma conseguiremos fazer a associação correta. Como fizemos com as habilidades, criaremos uma nova pasta em `resources`, chamada `nota` e, nela, o arquivo `cadastrar.html`, com o seguinte código:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" >
<head>
  <meta charset="UTF-8" />
  <link type="text/css" rel="stylesheet" href="../../materialize/css/materialize.min.css" media=
  <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet" />
  <title>EscolAlura</title>
  <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.1.min.js"></script>
</head>
<body class="grey lighten-3">
  <div id="formularioEdicao" class="container">
    <h3 class="main-title center">Cadastrar Nota para <span th:text="${aluno.nome}"></span></h3:
    <div class="row">
      <form class="col s12" action="#" th:action="@{'/nota/salvar/' + ${aluno.id}}" th:object="$
        <div class="section">
          <h5>Notas</h5>
          <div class="row">
            <div class="input-field col s12">
              <input id="valor" type="number" class="validate" th:value="${valor}" name="valor"
              <label for="valor">Nota</label>
            </div>
```

```
          </div>
        </div>


        <div class="row">
          <div class="input-field col s12 center">
            <button class="btn waves-effect waves-light" type="submit" name="action">Salvar Nota
            </button>
          </div>
        </div>
      </form>
    </div>

  </div> <!-- Fim do formulario de edicao -->

  <script type="text/javascript" src="../../materialize/js/materialize.min.js"></script>
</body>
</html>
```

Podemos testar e ver que o formulário de notas já pode ser visualizado, porém, ainda não funciona. Isso ocorre por precisarmos codificar a lógica que salva as notas dos alunos no banco. Na classe `NotaController`, criaremos o método `salvar`, que fará esta tarefa para nós. O código é muito parecido com o de habilidades, veja só:

```
@PostMapping("/nota/salvar/{id}")
public String salvar(@PathVariable String id, @ModelAttribute Nota nota) {
  Aluno aluno = repository.obterAlunoPor(id);
  repository.salvar(aluno.adicionar(aluno, nota));
  return "redirect:/aluno/listar";
}
```

Precisaremos criar também o método `adicionar` na classe `aluno`, recebendo como argumento o aluno e a nota, em vez de habilidade:

```
public Aluno adicionar(Aluno aluno, Nota nota) {
  List<Nota> notas = aluno.getNotas();
  notas.add(nota);
  aluno.setNotas(notas);
  return aluno;
}
```

É necessário nos certificarmos de que o método `getNotas` não retorne nulo, e sim uma lista vazia, caso não exista nenhuma nota para o aluno. No método `getNotas`, faremos esta validação com o `if`.

```
public List<Nota> getNotas() {
  if(notas == null) {
    notas = new ArrayList<Nota>();
  }
  return notas;
}
```

Ao testarmos novamente, lembre-se de sempre verificar os resultados diretamente no banco para confirmar! Cadastraremos a nota `9` para a Julia, cuja habilidade de inglês cadastramos quando a criamos. Porém, no banco temos um resultado não esperado.

Lembre-se de que para o cadastro de notas é utilizada a terceira opção na coluna de ações da listagem.

```
{
  "_id" : ObjectId("599e4207a8a8132b5db1995a"),
  "nome" : "Julia",
  "data_nascimento" : ISODate("2017-04-11T03:00:00Z"),
  "curso" : {
    "nome" : "Sistemas de Informação"
  },
  "habilidades" : [ ]
}
```

Não só a nota da Júlia não foi cadastrada como a habilidade de inglês intermediário foi removida. Como assim? O problema é que as notas não foram salvas pois não instruímos ao Mongo como fazê-lo em nosso *codec*. Já o problema das habilidades ocorre porque o Mongo não soube decodificá-las para objetos Java, então, como o aluno vem do banco sem elas, não haverá habilidades a serem escritas quando o aluno voltar ao banco, e é neste momento que a sobrescrita acontece.

Para resolvermos estes dois problemas, precisaremos inicialmente implementar no *codec* a lógica de *encode* e *decode* das notas. Depois, é preciso implementar o *decode* das habilidades. Para fazer o *encode* das notas teremos um código bem parecido com o de habilidades, note:

```
// código omitido
List<Nota> notas = aluno.getNotas();

if(notas != null) {
  List<Double> notasParaSalvar = new ArrayList<>();
  for (Nota nota : notas) {
    notasParaSalvar.add(nota.getValor());
  }
  document.put("notas", notasParaSalvar);
}
// código omitido
```

Note que este trecho de código deve ser escrito no método `encode` da classe `AlunoCodec`. Feito este passo, codifica-se as notas da forma correta. Agora, poderemos decodificar as notas e habilidades antes de retornar o aluno. Assim:

```
// código omitido
List<Double> notas = (List<Double>) document.get("notas");
if (notas != null) {
  List<Nota> notasDoAluno = new ArrayList<>();
  for (Double nota : notas) {
    notasDoAluno.add(new Nota(nota));
    aluno.setNotas(notasDoAluno);
  }
}
```

```
List<Document> habilidades = (List<Document>) document.get("habilidades");
if(habilidades != null) {
  List<Habilidade> habilidadesDoAluno = new ArrayList<>();
  for (Document documentHabilidade : habilidades) {
    habilidadesDoAluno.add(new Habilidade(documentHabilidade.getString("nome"), documentHabilida
  }
  aluno.setHabilidades(habilidadesDoAluno);
}
// código omitido
```

Atenção ao fato de que as notas estão sendo convertidas diretamente para uma lista de *Doubles*, pois são valores diretamente numéricos. Isto não ocorre com as habilidades, que são objetos com dois atributos: nome e nível. Talvez você já tenha percebido, mas este código não funciona, já que nenhuma das duas classes (`Habilidade` e `Nota`) possuem construtores que recebem valores desta maneira.

Por isso precisaremos criá-los também, deixando um construtor vazio para que o *String* consiga criar os objetos dos formulários, por exemplo.

Na classe `Nota` adicionaremos os construtores:

```
public class Nota {
  private Double valor;

  public Nota() {
  }

  public Nota(Double valor) {
    this.valor = valor;
  }
  //código omitido
}
```

O mesmo vale para a classe `Habilidade` :

```
public class Habilidade {
  private String nome;

  private String nivel;

  public Habilidade() {
  }

  public Habilidade(String nome, String nivel) {
    this.nome = nome;
    this.nivel = nivel;
  }

 //código omitido
}
```

Vamos testar novamente o cadastro de notas, bem como o de habilidades, para verificarmos se as notas são cadastradas de fato, e se as habilidades não são sobrescritas. Recadastraremos a habilidade de inglês intermediário para a Julia e lhe

daremos também uma nota, `8` desta vez. No banco teremos:

```
{
  "_id" : ObjectId("599e4207a8a8132b5db1995a"),
  "nome" : "Julia",
  "data_nascimento" : ISODate("2017-04-11T03:00:00Z"),
  "curso" : {
    "nome" : "Sistemas de Informacao"
  },
  "habilidades" : [
    {
      "nome" : "Ingles",
      "nivel" : "Intermediario"
    }
  ],
  "notas" : [
    8
  ]
}
```

Agora sim, tudo funciona como esperado!