01

Complementando o aluno

Transcrição

O cadastro e a listagem de alunos já funcionam, mas ainda precisaremos cadastrar as habilidades e notas dos alunos - será que tudo está funcionando, mesmo? Experimente abrir novamente o cadastro de alunos; teremos um erro.

```
java.lang.NoSuchMethodException: br.com.alura.escolalura.models.Curso.<init>()
```

O problema é que para criarmos o objeto `aluno` para popularmos a listagem, anteriormente usávamos um construtor vazio, e depois o sobrescrevíamos no momento de decodificar o objeto no *codec*. A solução é simples: criar um construtor vazio.

```
public class Curso {
  private String nome;
  public Curso(){

  }
  // código omitido
}
```

> Por padrão, todas as classes possuem um construtor vazio, porém, quando escrevemos um construtor explicitamente, o vazio passa a não existir mais, a não ser que o deixemos explícito também.

Podemos completar o cadastro dos nossos alunos e, para as habilidades de um determinado aluno, precisaremos saber de que aluno se trata. Por este motivo utilizaremos a listagem construída na aula anterior. Ela possui três ações para cada aluno: visualizar, cadastrar habilidades e notas, as quais se encontram na coluna `ações` , respectivamente, da esquerda para a direita.

Precisaremos criar um novo template a ser carregado ao respectivo aluno clicado na listagem, de maneira a editarmos um aluno específico. Segue o HTML deste template:

```
<!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="screen,projection" />
<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 Habilidade para <span th:text="${aluno.nome}"></span>
    <div class="row">
      <form class="col s12" action="#" th:action="@{'/habilidade/salvar/' + ${aluno.id}}" th:obj
        <div class="section">
          <h5>Habilidades</h5>
          <div class="row">
            <div class="input-field col s12">
              <input id="nome" type="text" class="validate" th:value="${habilidade.nome}" name='
                    <label for="nome">Nome</label>
            </div>
          </div>
          <div class="row">
            <div class="input-field col s12">
              <input id="nivel" type="text" class="validate" th:value="${habilidade.nivel}" name
                    <label for="nivel">Nível</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 Hab
              </button>
          </div>
        </div>
      </form>
    </div>

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

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

</body>
</html>
```

Por se tratar de um novo domínio da aplicação, faremos esse trabalho separado do aluno em si. Criaremos uma nova pasta em `resources` chamada `habilidade` e o template se chamará `cadastrar.html` . O processo de cadastro de habilidades requer suas próprias rotas, bem como seus próprios templates. Considerando isso, criaremos um *controller* de habilidades:

```
package br.com.alura.escolalura.controllers;


@Controller
public class HabilidadeController {

  @GetMapping("habilidade/cadastrar/{id}")
  public String cadastrar(@PathVariable String id, Model model){
    model.addAttribute("aluno", aluno);
    model.addAttribute("habilidade", new Habilidade());
    return "habilidade/cadastrar";
  }

}
```

Identificaremos o aluno com que estamos lidando no cadastro de habilidades, e passamos seu *id* como parâmetro no mapeamento da rota, recuperando-o com a anotação `@PathVariable`. Note também que precisaremos tanto do aluno quanto do objeto `habilidade`. Isso porque o mesmo é usado na construção do formulário. Como vamos trazer o aluno para este template?

Para buscar um aluno em nossa coleção, precisaremos de um novo método em `AlunoRepository`, que se chamará `obterAlunoPor` e receberá o *id* do aluno, a partir do qual se fará a busca e o retornará.

```
public Aluno obterAlunoPor(String id){
  criarConexao();
  MongoCollection<Aluno> alunos = this.bancoDeDados.getCollection("alunos", Aluno.class);
  Aluno aluno = alunos.find(Filters.eq("_id", new ObjectId(id))).first();
  return aluno;
}
```

Lembre-se de que precisaremos chamar o método `criarConexao` sempre. Na primeira aula vimos o uso dos filtros, então não há nenhuma novidade neste código. Observe apenas que, além de fazer a busca com o filtro, estamos retornando apenas seu primeiro resultado.

Feito o método de pesquisa por *id*, pode-se injetar nosso repositório no *controller* de habilidades, utilizando o método para termos o aluno pronto para ser enviando ao template.

```
@Controller
public class HabilidadeController {

  @Autowired
  private AlunoRepository repositorio;

  @GetMapping("habilidade/cadastrar/{id}")
  public String cadastrar(@PathVariable String id, Model model){
    Aluno aluno = repositorio.obterAlunoPor(id);
    model.addAttribute("aluno", aluno);
    model.addAttribute("habilidade", new Habilidade());
    return "habilidade/cadastrar";
  }

}
```

Indo direto à listagem dos alunos e clicando no segundo ícone da coluna de ações, somos direcionados ao cadastro de habilidades de cada aluno. A imagem abaixo ilustra o formulário de cadastro de habilidades da Julia!