02

Salvando habilidades do aluno

Transcrição

O processo de cadastro das habilidades do aluno é muito simples. Abrimos o formulário, preenchemos os campos de nome e nível da habilidade e clicamos em *salvar habilidade*. Em nosso *controller*, o que precisamos inicialmente é mapear o *POST* deste formulário e capturar o *id* do aluno juntamente com os dados da habilidade. Considerando isso criaremos o método `salvar` na classe `HabilidadeController`.

```
@PostMapping("habilidade/salvar/{id}")
public String salvar(@PathVariable String id, @ModelAttribute Habilidade habilidade){

  return "redirect: /aluno/listar";
}
```

Note que como retorno, estamos fazendo um redirecionamento para a listagem dos alunos. Nosso próximo passo é, obter o aluno pelo *id* e adicionar essa habilidade recebida pelo método em suas habilidades. Algo como descrito no código abaixo.

```
@PostMapping("habilidade/salvar/{id}")
public String salvar(@PathVariable String id, @ModelAttribute Habilidade habilidade){
  Aluno aluno = repositorio.obterAlunoPor(id);
  aluno.adiciona(aluno, habilidade);
  return "redirect: /aluno/listar";
}
```

Como já deve ter percebido, o método `adiciona` não existe no nosso modelo de alunos. Vamos criá-lo então! Devemos nele obter as habilidades do aluno, adicionar a nova e retornar este aluno para ser salvo no banco de dados.

```
public Aluno adiciona(Aluno aluno, Habilidade habilidade) {
  List<Habilidade> habilidades = aluno.getHabilidades();
  habilidades.add(habilidade);
  aluno.setHabilidades(habilidades);
  return aluno;
}
```

Agora podemos passar o retorno do método `adiciona` direto para o método `salvar` do objeto `repositorio`.

```
@PostMapping("habilidade/salvar/{id}")
  public String salvar(@PathVariable String id, @ModelAttribute Habilidade habilidade){
    Aluno aluno = repositorio.obterAlunoPor(id);

    repositorio.salvar(aluno.adiciona(aluno, habilidade));

    return "redirect:/aluno/listar";
  }
```

Se você se recordar bem, vai notar que o método salvar apenas insere um novo registro, se o deixarmos como está estaremos duplicando o registro de aluno para cada habilidade cadastrada. Precisamos criar alguma validação para que ao invés de cadastrar um novo aluno, atualize o aluno.

```
public void salvar(Aluno aluno){
  criarConexao();
  MongoCollection<Aluno> alunos = this.bancoDeDados.getCollection("alunos", Aluno.class);

  if(aluno.getId() == null){
    alunos.insertOne(aluno);
  }else{
    alunos.updateOne(Filters.eq("_id", aluno.getId()), new Document("$set", aluno));
  }

}
```

A validação não passa de um simples `if` que verifica: se o *id* do aluno não existe, este é um aluno novo e portando deverá ser adicionado a coleção de alunos. Caso contrário, utilizamos o método `updateOne` filtrando o aluno pelo *id* e indicando que esta é uma atualização parcial do aluno.

Um ponto que deixamos passar, mas que fará sentido fazer uma nova validação é: E se o atributo de habilidades do aluno for nulo? Teremos uma exceção por que estaremos tentando adicionar uma habilidade ao nulo. Por isso, no método `getHabilidades` da classe `Aluno` faremos a seguinte validação.

```
public List<Habilidade> getHabilidades() {
  if(habilidades == null){
    habilidades = new ArrayList<Habilidade>();
  }
  return habilidades;
}
```

Caso `habilidades` seja nulo, criamos uma listagem vazia. Caso não, ela será retornada normalmente. Testamos cadastrar uma nova habilidade para a Júlia e tudo parece funcionar normalmente, mas ao verificar o registro da Júlia em nosso banco de dados temos:

```
{
  "_id" : ObjectId("5998946e161a4b2792ed78fc"),
  "nome" : "Julia",
  "data_nascimento" : ISODate("2017-08-13T03:00:00Z"),
  "curso" : {
    "nome" : "Administração"
  }
}
```

Onde estão as habilidades da Júlia? O fato é que o nosso *codec* não sabe tratar o atributo `habilidades`. Isso indica que, para cada novo atributo que adicionamos ao nosso modelo, precisamos ensinar ao Mongo como tratá-lo através do nosso *codec*.

O que faremos no caso das habilidades é simples: verificaremos se esse atributo não é nulo, criaremos uma listagem de documentos para cada habilidade e adicionaremos essa listagem de documentos no campo `habilidades` do nosso aluno. No método `encode` da classe `AlunoCodec` ficará assim:

```
public void encode(BsonWriter writer, Aluno aluno, EncoderContext encoder) {
  ObjectId id = aluno.getId();
  String nome = aluno.getNome();
  Date dataNascimento = aluno.getDataNascimento();
  Curso curso = aluno.getCurso();
  List<Habilidade> habilidades = aluno.getHabilidades();

  Document documento = new Document();
  documento.put("_id", id);
  documento.put("nome", nome);
  documento.put("data_nascimento", dataNascimento);
  documento.put("curso", new Document("nome", curso.getNome()));

  if(habilidades != null){
    List<Document> habilidadesDocument = new ArrayList<>();
    for (Habilidade habilidade : habilidades) {
      habilidadesDocument.add(new Document("nome", habilidade.getNome())
          .append("nivel", habilidade.getNivel()));
    }
    documento.put("habilidades", habilidadesDocument);
  }

  codec.encode(writer, documento, encoder);
}
```

Se testarmos novamente e verificarmos nossa base de dados teremos uma nova habilidade para a Júlia:

```
{
  "_id" : ObjectId("5998946e161a4b2792ed78fc"),
  "nome" : "Julia",
  "data_nascimento" : ISODate("2017-08-13T03:00:00Z"),
  "curso" : {
    "nome" : "Administração"
  },
  "habilidades" : [
    {
      "nome" : "Inglês",
      "nivel" : "Intermediário"
    }
  ]
}
```