04

Criando um repository

Transcrição

Já vimos como receber um objeto `aluno` em nosso *controller*, então codificaremos a lógica necessária para salvar este aluno no banco de dados.

Criaremos a classe `AlunoRepository` no pacote `br.com.alura.escolalura.repositories`, e esta classe terá o método `salvar`, a receber o objeto `aluno`, abrindo a conexão e persistindo-o.

```
public class AlunoRepository {

  public void salvar(Aluno aluno){
    MongoClient cliente = new MongoClient();
    MongoDatabase bancoDeDados = cliente.getDatabase("test");
    MongoCollection<Aluno> alunos = bancoDeDados.getCollection("alunos", Aluno.class);
    alunos.insertOne(aluno);
  }

}
```

Como o método `salvar` não possui retorno, o mesmo é denominado `void`. Este código é bem semelhante ao que já fizemos nas outras vezes, porém, perceba que estamos indicando o tipo de retorno da coleção do MongoDB no segundo argumento do método `getCollection`. O padrão do Mongo é trabalhar com `Document`, porém estamos deixando o tipo explícito, e queremos uma coleção de `Aluno`.

Como esta classe será nosso repositório de alunos, precisaremos indicar ao Spring que é um `Repository`, anotando-a como tal. A classe `AlunoRepository`, completa, fica assim:

```
@Repository
public class AlunoRepository {

  public void salvar(Aluno aluno){
    MongoClient cliente = new MongoClient();
    MongoDatabase bancoDeDados = cliente.getDatabase("test");
    MongoCollection<Aluno> alunos = bancoDeDados.getCollection("alunos", Aluno.class);
    alunos.insertOne(aluno);
  }

}
```

O último passo para testarmos se tudo funciona como esperado é fazer a injeção deste repositório no *controller*, utilizando-o no método `salvar` da classe `AlunoController`. Este trecho de código é demonstrado abaixo:

```
@Controller
public class AlunoController {

  @Autowired
```

```
    private AlunoRepository repositorio;

    // código omitido

    @PostMapping("/aluno/salvar")
    public String salvar(@ModelAttribute Aluno aluno){
      repositorio.salvar(aluno); // salvando o aluno
      System.out.println(aluno);
      return "redirect:/";
    }
  }
```

Como queremos que o Spring faça a injeção do repositório automaticamente, utilizaremos a anotação `@Autowired` e, no método `salvar`, usaremos este objeto. Podemos testar!

Note que ao tentarmos cadastrar um aluno, teremos um erro, e no console haverá a mensagem de erro indicando que precisamos de um **codec** para a classe `aluno`.

```
Can't find a codec for class br.com.alura.escolalura.models.Aluno
```

Codec? Como assim? A primeira lembrança que temos quando pensamos nisso é quando precisamos assistir vídeos ou ouvir músicas em alguma máquina nova. Os *codecs* funcionam como tradutores, de um tipo de codificação para outro. O que acontece no Mongo é que ele sabe muito bem trabalhar com documentos, mas não com alunos.

Como deixamos explícito no código que queremos um retorno específico de uma coleção de alunos no método `salvar` da classe `AlunoRepository`, é preciso criar este *codec* para que o Mongo consiga traduzir um documento "aluno", para um objeto `aluno`.

> Sobre a camada *service*, é comum que se crie uma camada de serviço nas aplicações, a fim de que os objetos passem por ela antes de serem persistidos na base de dados. Neste caso, como não há regras de negócio, não precisaremos necessariamente da camada de serviço, podendo chamar diretamente a camada de acesso a dados. Isso também evita que criemos modelos anêmicos, uma camada de serviço sem regras de negócio, apenas para chamarmos a camada de dados.

Para a criação do *codec* de alunos, há duas bibliotecas principais, sendo a primeira do próprio Mongo, chamada **Morphia** (https://mongodb.github.io/morphia/), e a segunda é **MongoJack** (http://mongojack.org/). Porém, também é possível criarmos nosso próprio *codec*, de forma a informarmos diretamente ao Mongo como os nossos objetos `aluno` s devem ser tratados.