



INSTITUTO FEDERAL
Triângulo Mineiro
Campus Uberlândia Centro

Projeto Backend

Microserviços e NoSQL

Relacionamentos no MongoDB

Prof. Lucas Montanheiro
lucasmontanheiro@iftm.edu.br

Criando um novo projeto

- Vamos criar um novo projeto chamado relacionamentos-mongodb.
- Use CTRL + ALT + P para acessar a paleta de comandos e procure por **Java: Create Java Project**
- Escolha um projeto do tipo **Maven**
- Não selecione **Archetype**.

Criando um novo projeto

- Utilize como Group Id **com.MEUNOME.mongodb.relacionamentos**
- Utilize como Artifact Id **relacionamentos**
- Use CTRL + ALT + P para acessar a paleta de comandos e procure por **Maven: Add a dependency** e adicione:
 - **spring-boot-starter-data-mongodb**
 - **spring-boot-starter-web**
 - **lombok**

Criando um novo projeto

- Vamos começar criando o **application.yml** dentro da pasta **resources** com o acesso ao MongoDB Atlas:

```
application.yml x
relacionamentos > src > main > resources > application.yml
1  spring:
2    data:
3      mongodb:
4        uri: mongodb+srv://lucasmontanheiro:*****@cluster0.7mli0.mongodb.net/
5        database: relacionamentos
```

Criando um novo projeto

- Vamos parametrizar nossa classe **Main.java** para utilizar **Spring Boot** e também o **MongoDB**:

```
relacionamentos > src > main > java > com > montanha > mongodb > relacionamentos > J Main.java > ...  
1  package com.montanha.mongodb.relacionamentos;  
2  
3  import org.springframework.boot.SpringApplication;  
4  import org.springframework.boot.autoconfigure.SpringBootApplication;  
5  import org.springframework.data.mongodb.repository.config.EnableMongoRepositories;  
6  
7  @SpringBootApplication  
8  @EnableMongoRepositories  
9  public class Main {  
10     Run | Debug  
11     public static void main(String[] args) {  
12         SpringApplication.run(Main.class, args);  
13     }  
14 }
```

Criando um novo projeto

- Caso não tenha conseguido criar o projeto, faça um clone da base disponível no GitHub:
 - <https://github.com/Montanheiro/projeto-ms-java-mongodb-iftm2024/tree/main/03-relacionamentos-mongodb>

Criando o models

- Vamos criar nossa pasta **models** para que possamos começar a criar nosso banco.
- Começamos criando as classes **Perfil** e **Usuario**, para que possamos fazer um relacionamento **Um-para-Um** (*One-to-One*).
- Onde um **Usuario** sempre tem um **Perfil**.

Criando o models

```
relacionamentos > src > main > java > com > montanha > mongodb > relacionamentos > models > J Perfil.java > ...  
1  package com.montanha.mongodb.relacionamentos.models;  
2  
3  import org.springframework.data.annotation.Id;  
4  import org.springframework.data.mongodb.core.mapping.Document;  
5  
6  import lombok.Data;  
7  
8  @Data  
9  @Document(collection = "perfis")  
10 public class Perfil {  
11     @Id  
12     private String id;  
13     private String bio;  
14     private String avatarUrl;  
15 }
```


Criando o models

relacionamentos > src > main > java > com > montanha > mongodb > relacionamentos > models >  Usuario.java > ...

```
1 package com.montanha.mongodb.relacionamentos.models;
2
3 import org.springframework.data.annotation.Id;
4 import org.springframework.data.mongodb.core.mapping.Document;
5 import org.springframework.data.mongodb.core.mapping.DBRef;
6
7 import lombok.Data;
8
9 @Data
10 @Document(collection = "usuarios")
11 public class Usuario {
12     @Id
13     private String id;
14     private String nome;
15
16     @DBRef
17     private Perfil perfil;
18 }
```

Anotação @DBRef

- O **@DBRef** é uma anotação usada no Spring Data MongoDB para criar referências entre documentos armazenados no MongoDB.
- Ele permite que um documento se refira a outro documento, simulando um relacionamento como os que existem em bancos de dados relacionais (como *foreign key*).

Anotação @DBRef

- Ao usar **@DBRef**, em vez de embutir os dados de outro documento dentro do documento principal, você armazena apenas uma referência ao documento relacionado, usando seu **_id**.
- Em nosso caso teremos duas coleções no MongoDB: **usuarios** e **perfils**.
- Um documento em **usuarios** pode referenciar um documento em **perfils** usando **@DBRef**.

Anotação @DBRef

- Em muitos casos, é preferível **embutir** o documento referenciado diretamente no documento principal (utilizando o conceito de **desnormalização**) para melhorar a performance e simplificar consultas.
- Porém casos como quando os dados referenciados são grandes e frequentemente reutilizados por outros documentos.
- E também quando há necessidade de simular relacionamentos complexos, como em modelos fortemente normalizados, podemos utilizar **@DBRef**.

Criando o repository

- Agora vamos criar a pasta **repositories** dentro da pasta vamos criar as interfaces **UsuarioRepository.java** e a **PerfilRepository.java**.

```
relacionamentos > src > main > java > com > montanha > mongodb > relacionamentos > repositories > PerfilRepository.java > ...
1 package com.montanha.mongodb.relacionamentos.repositories;
2
3 import org.springframework.data.mongodb.repository.MongoRepository;
4 import com.montanha.mongodb.relacionamentos.models.*;
5
6 public interface PerfilRepository extends MongoRepository<Perfil, String> {
7 }
```

```
relacionamentos > src > main > java > com > montanha > mongodb > relacionamentos > repositories > UsuarioRepository.java > .
1 package com.montanha.mongodb.relacionamentos.repositories;
2
3 import org.springframework.data.mongodb.repository.MongoRepository;
4 import com.montanha.mongodb.relacionamentos.models.*;
5
6 public interface UsuarioRepository extends MongoRepository<Usuario, String> {
7 }
```

Criando o repository


- Já podemos testar implementando apenas o **Controller**, sem a utilização de Service para ficar mais simples.

```
relacionamentos > src > main > java > com > montanha > mongodb > relacionamentos > controllers > J UsuarioController.java > ...  
1 package com.montanha.mongodb.relacionamentos.controllers;  
2  
3 import com.montanha.mongodb.relacionamentos.models.Perfil;  
4 import com.montanha.mongodb.relacionamentos.models.Usuario;  
5 import com.montanha.mongodb.relacionamentos.repositories.UsuarioRepository;  
6 import com.montanha.mongodb.relacionamentos.repositories.PerfilRepository;  
7 import org.springframework.beans.factory.annotation.Autowired;  
8 import org.springframework.web.bind.annotation.*;  
9  
10 import java.util.List;  
11  
12 @RestController  
13 @RequestMapping("/usuarios")  
14 public class UsuarioController {  
15     @Autowired  
16     private UsuarioRepository usuarioRepository;  
17  
18     @Autowired  
19     private PerfilRepository perfilRepository;  
20  
21     @GetMapping  
22     public List<Usuario> getAll() {  
23         return usuarioRepository.findAll();  
24     }  
25  
26     @PostMapping  
27     public Usuario create(@RequestBody Usuario usuario) {  
28         if (usuario.getPerfil() != null && usuario.getPerfil().getId() == null) {  
29             Perfil perfilSalvo = perfilRepository.save(usuario.getPerfil());  
30             usuario.setPerfil(perfilSalvo);  
31         }  
32         return usuarioRepository.save(usuario);  
33     }  
34 }
```

Testando o endpoint

- Ao fazermos um POST no endpoint criado e olharmos no banco de dados, vamos ter duas novas collections:

▼  relacionamentos

 perfis

 usuarios

Testando o endpoint

- Quando verificamos o que existe nas **collections**:

```
_id: ObjectId('673ba051cd0f4631f64e2097')  
bio : "Engenheiro de Software"  
avatarUrl : "https://meuavatar.com/avatar123.jpg"  
_class : "com.montanha.mongodb.relacionamentos.models.Perfil"
```

```
_id: ObjectId('673ba051cd0f4631f64e2098')  
nome : "João Silva"  
perfil : DBRef('perfis', '673ba051cd0f4631f64e2097')  
_class : "com.montanha.mongodb.relacionamentos.models.Usuario"
```


Testando o endpoint

- Ao fazermos o **GET** no **endpoint** criado temos os dados de uma forma melhor estruturada:

```
1  [
2    {
3      "id": "673ba051cd0f4631f64e2098",
4      "nome": "João Silva",
5      "perfil": {
6        "id": "673ba051cd0f4631f64e2097",
7        "bio": "Engenheiro de Software",
8        "avatarUrl": "https://meuavatar.com/avatar123.jpg"
9      }
10   }
11 ]
12
```

Um para Muitos

- Vamos fazer um exemplo de Um para Muitos, onde um **Usuario** tem várias **Postagens**.
- Vamos começar criando o **models** de **Postagem**.

Um para Muitos

relacionamentos > src > main > java > com > montanha > mongodb > relacionamentos > models >  Postagem.java > ...

```
1  package com.montanha.mongodb.relacionamentos.models;
2
3  import lombok.Data;
4  import org.springframework.data.annotation.Id;
5  import org.springframework.data.mongodb.core.mapping.Document;
6
7  @Data
8  @Document(collection = "postagens")
9  public class Postagem {
10     @Id
11     private String id;
12     private String titulo;
13     private String conteudo;
14 }
```

Um para Muitos

- Na **model** de **Usuario**, vamos inserir uma lista de **postagens**.

```
11  @Data
12  @Document(collection = "usuarios")
13  public class Usuario {
14      @Id
15      private String id;
16      private String nome;
17
18      @DBRef
19      private Perfil perfil;
20
21      @DBRef
22      private List<Postagem> postagens;
23
24  }
```

Muitos para Muitos

- Para fazer o relacionamento de muitos para muitos, vamos ter duas novas models, são: **Estudante** e **Curso**.
- Onde ambos tem listas de ambos.

Muitos para Muitos

relacionamentos > src > main > java > com > montanha > mongodb > relacionamentos > models > J Estudante.java > .

```
1 package com.montanha.mongodb.relacionamentos.models;
2
3 import lombok.Data;
4 import org.springframework.data.annotation.Id;
5 import org.springframework.data.mongodb.core.mapping.Document;
6 import org.springframework.data.mongodb.core.mapping.DBRef;
7 import java.util.List;
8
9 @Data
10 @Document(collection = "estudantes")
11 public class Estudante {
12     @Id
13     private String id;
14     private String nome;
15
16     @DBRef
17     private List<Curso> cursos;
18 }
```

Muitos para Muitos

relacionamentos > src > main > java > com > montanha > mongodb > relacionamentos > models > Curso.java > ...

```
1  package com.montanha.mongodb.relacionamentos.models;
2
3  import lombok.Data;
4  import org.springframework.data.annotation.Id;
5  import org.springframework.data.mongodb.core.mapping.Document;
6  import org.springframework.data.mongodb.core.mapping.DBRef;
7  import java.util.List;
8
9  @Data
10 @Document(collection = "cursos")
11 public class Curso {
12     @Id
13     private String id;
14     private String nome;
15
16     @DBRef
17     private List<Estudante> estudantes;
18 }
```

Finalizar Implementação

- Termine a implementação dos **Controllers** e **Repositories** e realize os testes com o Postman.
- Crie os seguintes endpoints:
 - POST /usuarios - adiciona um usuário
 - GET /usuarios/{id} - retorna apenas o ID selecionado
 - GET /usuarios - retorna todos os usuários, com seus relacionamentos
 - PUT /usuarios/{id} - atualiza um usuário
 - DELETE /usuario/{id} - deleta um usuário

Finalizar Implementação

- POST /perfis - adiciona um
- GET /perfis - retorna todos
- GET /perfis/{id} - retorna apenas o ID selecionado
- PUT /perfis/{id} - atualiza um
- DELETE /perfis/{id} - deleta um

Finalizar Implementação

- POST /postagens - adiciona um
- GET /postagens - retorna todos
- GET /postagens/{id} - retorna apenas o ID selecionado
- PUT /postagens/{id} - atualiza um
- DELETE /postagens/{id} - deleta um

Finalizar Implementação

- POST /estudantes - adiciona um
- GET /estudantes - retorna todos
- GET /estudantes/{id} - retorna apenas o ID selecionado
- PUT /estudantes/{id} - atualiza um
- DELETE /estudantes/{id} - deleta um

Finalizar Implementação

- POST /cursos - adiciona um
- GET /cursos - retorna todos
- GET /cursos/{id} - retorna apenas o ID selecionado
- PUT /cursos/{id} - atualiza um
- DELETE /cursos/{id} - deleta um