



INSTITUTO FEDERAL
Triângulo Mineiro
Campus Uberlândia Centro

Projeto Backend

Microserviços e NoSQL

Integrando projeto ao MongoDB

Prof. Lucas Montanheiro
lucasmontanheiro@iftm.edu.br

Comandos para subir o MongoDB

- Vamos utilizar o MongoDB Atlas.
- Pegue a URL e senha de conexão ao MongoDB para seguirmos.

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**
- Utilize como Artifact Id **userapi**
- 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**

Configurando o application.properties

- Adicione as informações do banco de dados MongoDB no:
- resources > application.yml
- Caso não tenha o arquivo .yml, crie e apague o .properties

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

Primeiros passos com MongoDB no Java

- Em nossa classe principal do projeto, precisamos adicionar a notação **@EnableMongoRepositories**, que automaticamente será feita a importação do Spring Boot Data com MongoDB.

```
src > main > java > com > montanha > user_api > UserApiApplication.java > Language Support for Java(TM) by Red Hat > UserApiApplication
1  package com.montanha.user_api;
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 UserApiApplication {
10
11      Run | Debug
12      public static void main(String[] args) {
13          SpringApplication.run(UserApiApplication.class, args);
14      }
15  }
```

Camada de Dados (Repository)

- Vamos criar uma nova pasta no main do projeto chamada **models** e mover a pasta **dto** para dentro dela.
- Agora sempre que tivermos uma classe DTOs, também será necessário criar uma entidade, que é o objeto que possui exatamente a mesma estrutura do banco de dados.
- Essas classes são anotadas com **@Document**, indicando que elas representam um documento dentro de uma coleção no MongoDB.

Camada de Dados (Repository)

- Nessa classe, o ID da entidade é marcado com a anotação **@Id**, que indica que o campo é o identificador único do documento.
- No MongoDB, o campo **_id** é gerado automaticamente e, por padrão, é do tipo **ObjectId**, porém para evitar conversões, vamos utilizar do tipo **String**.
- Se necessário, você pode configurar manualmente o tipo do campo ou gerar IDs personalizados.

Camada de Dados (Repository)

- Além disso, os outros atributos podem ser marcados com a anotação **@Field**, que possui propriedades como se o campo é obrigatório ou não, o nome do campo no documento, e outras configurações.
- Assim como na classe UserDTO, iremos adicionar as anotações do **Lombok** na classe User para simplificar a criação de **getters**, **setters** e **construtores**.
- Vamos criar a **User.class** dentro de **models**:

Classe User em models

```
src > main > java > com > montanha > user_api > models > J User.java > ...
1 package com.montanha.user_api.models;
2
3 import java.time.LocalDateTime;
4
5 import org.springframework.data.annotation.Id;
6 import org.springframework.data.mongodb.core.mapping.Document;
7 import org.springframework.data.mongodb.core.mapping.Field;
8 import com.montanha.user_api.models.dto.UserDTO;
9 import lombok.AllArgsConstructor;
10 import lombok.Data;
11 import lombok.NoArgsConstructor;
12
13 @Data
14 @NoArgsConstructor
15 @AllArgsConstructor
16 @Document(collection = "user")
17 public class User {
18
19     @Id
20     private String id;
21     @Field("nome")
22     private String nome;
23     @Field("cpf")
24     private String cpf;
25     @Field("endereco")
26     private String endereco;
27     @Field("email")
28     private String email;
29     @Field("telefone")
30     private String telefone;
31     @Field("dataCadastro")
32     private LocalDateTime dataCadastro;
33
34     public static User convert(UserDTO userDTO) {
35         User user = new User();
36         user.setName(userDTO.getName());
37         user.setEndereco(userDTO.getEndereco());
38         user.setCpf(userDTO.getCpf());
39         user.setEmail(userDTO.getEmail());
40         user.setTelefone(userDTO.getTelefone());
41         user.setDataCadastro(userDTO.getDataCadastro());
42         return user;
43     }
44
45 }
```

Camada de Dados (Repository)

- O método `convert`, na classe **User**, é importante porque precisaremos converter instâncias da entidade **User** para instâncias da classe **UserDTO**.
- Isso será utilizado em diversos serviços, então é melhor ter um método que realiza essa operação do que ficar duplicando código em vários locais.
- A mesma coisa será necessária na classe **UserDTO** , como podemos ver a seguir:

Classe UserDTO em models > dto

```
src > main > java > com > montanha > user_api > models > dto > J UserDTO.java > ...
1 package com.montanha.user_api.models.dto;
2
3 import java.time.LocalDateTime;
4
5 import com.montanha.user_api.models.User;
6 import jakarta.validation.constraints.NotBlank;
7 import lombok.AllArgsConstructor;
8 import lombok.Getter;
9 import lombok.NoArgsConstructor;
10 import lombok.Setter;
11
12 @Getter
13 @Setter
14 @NoArgsConstructor
15 @AllArgsConstructor
16 public class UserDTO {
17
18     private String id;
19     @NotBlank(message = "Nome é obrigatório")
20     private String nome;
21     @NotBlank(message = "CPF é obrigatório")
22     private String cpf;
23     private String endereco;
24     @NotBlank(message = "E-mail é obrigatório")
25     private String email;
26     private String telefone;
27     private LocalDateTime dataCadastro;
28
29     public static UserDTO convert(User user) {
30         UserDTO userDTO = new UserDTO();
31         userDTO.setId(user.getId());
32         userDTO.setName(user.getName());
33         userDTO.setEndereco(user.getEndereco());
34         userDTO.setCpf(user.getCpf());
35         userDTO.setEmail(user.getEmail());
36         userDTO.setTelefone(user.getTelefone());
37         userDTO.setDataCadastro(user.getDataCadastro());
38         return userDTO;
39     }
40 }
```

Camada de Dados (Repository)

- Com o Spring Data MongoDB, agora vamos criar um repositório para a entidade criada.
- Primeiro vamos criar a pasta **repositories** e dentro da pasta a interface **UserRepository.java**.
- O repositório é uma interface anotada com **@Repository**, que também é um **bean** do Spring e será automaticamente instanciado na inicialização da aplicação.

Camada de Dados (Repository)

- Essa interface deve estender **MongoRepository**, passando a entidade **User** e o tipo do **ID**, que, no seu caso, é **String**.
- O código da interface **UserRepository** pode ser da seguinte forma:

```
src > main > java > com > montanha > user_api > repositories > J UserRepository.java > ...
1  package com.montanha.user_api.repositories;
2
3
4  import org.springframework.data.mongodb.repository.MongoRepository;
5  import org.springframework.stereotype.Repository;
6  import com.montanha.user_api.models.User;
7
8  @Repository
9  public interface UserRepository extends MongoRepository<User, String> {
10
11  }
```

Camada de Dados (Repository)

- Podemos adicionar também alguns métodos como **findByCpf** e **queryByNameLike**, algumas consultas podem ser criadas apenas com o nome do método.
- Esses métodos devem ter algumas palavras-chaves no nome como **find**, **and**, **or**, **like** e o nome do campo.
- Nos próximos serviços, criaremos algumas consultas mais complexas, e vamos ver mais dessas palavras-chaves.

Camada de Dados (Repository)

```
src > main > java > com > montanha > user_api > repositories > J UserRepository.java > ...
1  package com.montanha.user_api.repositories;
2
3  import java.util.List;
4
5  import org.springframework.data.mongodb.repository.MongoRepository;
6  import org.springframework.stereotype.Repository;
7  import com.montanha.user_api.models.User;
8
9  @Repository
10 public interface UserRepository extends MongoRepository<User, String> {
11
12     User findByCpf(String cpf);
13
14     List<User> queryByNomeLike(String nome);
15
16 }
17
```


Camada de Serviços (Service)

- O **controller** poderia chamar o repositório diretamente, porém a maioria das aplicações possui uma camada intermediária, chamada de **service**, que é onde ficam as regras de negócio da aplicação.
- Essas classes devem ser anotadas com **@Service** e normalmente são responsáveis por fazer chamadas ao repositório e também a outros serviços.

Camada de Serviços (Service)

- Vamos criar em nosso projeto a pasta **services** e a classe **UserService.java**.
- A listagem mostra um exemplo de uma classe **Service** que será utilizada para acessar os dados da collection **user**.

```
src > main > java > com > montanha > user_api > services > J UserService.java > ...
1  package com.montanha.user_api.services;
2
3  import java.util.List;
4  import java.util.stream.Collectors;
5  import org.springframework.stereotype.Service;
6  import com.montanha.user_api.models.dto.UserDTO;
7  import com.montanha.user_api.models.User;
8  import com.montanha.user_api.repositories.UserRepository;
9  import lombok.RequiredArgsConstructor;
10
11  @Service
12  @RequiredArgsConstructor
13  public class UserService {
14      private final UserRepository userRepository;
15
16      public List<UserDTO> getAll() {
17          List<User> usuarios = userRepository.findAll();
18          return usuarios.stream()
19              .map(UserDTO::convert)
20              .collect(Collectors.toList());
21      }
22  }
```

Camada de Serviços (Service)

- Na listagem, é possível ver que a classe foi anotada com **@Service**, indicando que uma instância dela será criada na criação da aplicação.
- A classe também possui um atributo do tipo **UserRepository**.
- Sem o Lombok é necessário usar o anotação **@Autowired**, que serve para fazer injeção de dependências.
- Porém, usando o Lombok, poderemos remover todas essas anotações e substituí-las pela anotação **@RequiredArgsConstructor**, que fica na classe.
- Com ela, evitamos ter que utilizar essa anotação **@Autowired** em classes que possuem a injeção de diversas dependências.

Camada de Serviços (Service)

Além disso, a classe possui o método **getAll**, que faz as seguintes operações:

- Chama o método **findAll**, do **UserRepository**, que retorna uma lista de usuários, sendo instâncias da entidade **User**;
- Transforma a lista em um **stream** e chama o método **map** para transformar a lista de entidades em uma lista de DTOs;
- Retorna a lista de DTOs.

Camada de Serviços (Service)

- Além do método **getAll**, adicionaremos mais seis métodos nesta classe: **findById**, **save**, **delete**, **findByCpf**, **queryByName** e **editUser**.
 - O **findById** busca um usuário por um id específico;
 - O **save** salva um usuário no banco de dados;
 - O **delete** exclui um usuário do banco de dados;
 - O **findByCpf** faz a busca de um usuário por seu CPF;
 - O **queryByName** faz uma busca pelo nome do usuário, mas, diferentemente da busca por id ou pelo CPF, a busca não será exata, mas sim pela inicial do nome passada no parâmetro. Por exemplo, se o parâmetro nome tiver valor **Mar** , a busca retornará pessoas com o nome Marcela, Marcelo ou Marcos.
 - O **editUser**, receberá o id de um usuário e um objeto da classe UserDTO. Com o id, será buscado um usuário no banco de dados e, caso ele exista, os dados desse usuário serão atualizados com o que foi recebido no objeto do tipo UserDTO; caso o usuário não exista, será retornado um erro de que o usuário não foi encontrado.

Implementar na UserService.java

```
21
22
23 public UserDTO findById(String userId) {
24     User usuario = userRepository.findById(userId)
25         .orElseThrow(() -> new RuntimeException("User not found"));
26     return UserDTO.convert(usuario);
27 }
28
29 public UserDTO save(UserDTO userDTO) {
30     userDTO.setDataCadastro(LocalDateTime.now());
31     User user = userRepository.save(User.convert(userDTO));
32     return UserDTO.convert(user);
33 }
34
35 public UserDTO delete(String userId) {
36     User user = userRepository
37         .findById(userId).orElseThrow(() -> new RuntimeException());
38     userRepository.delete(user);
39     return UserDTO.convert(user);
40 }
41
42 public UserDTO findByCpf(String cpf) {
43     User user = userRepository.findByCpf(cpf);
44     if (user != null) {
45         return UserDTO.convert(user);
46     }
47     return null;
48 }
49
50 public List<UserDTO> queryByName(String name) {
51     List<User> usuarios = userRepository.queryByNomeLike(name);
52     return usuarios
53         .stream()
54         .map(UserDTO::convert)
55         .collect(Collectors.toList());
56 }
57
58
59
60
```

```
60
61 public UserDTO editUser(String userId, UserDTO userDTO) {
62     User user = userRepository
63         .findById(userId).orElseThrow(() -> new RuntimeException());
64     if (userDTO.getEmail() != null &&
65         !user.getEmail().equals(userDTO.getEmail())) {
66         user.setEmail(userDTO.getEmail());
67     }
68     if (userDTO.getTelefone() != null &&
69         !user.getTelefone().equals(userDTO.getTelefone())) {
70         user.setTelefone(userDTO.getTelefone());
71     }
72     if (userDTO.getEndereco() != null &&
73         !user.getEndereco().equals(userDTO.getEndereco())) {
74         user.setEndereco(userDTO.getEndereco());
75     }
76     user = userRepository.save(user);
77     return UserDTO.convert(user);
78 }
79
80 public Page<UserDTO> getAllPage(Pageable page) {
81     Page<User> users = userRepository.findAll(page);
82     return users
83         .map(UserDTO::convert);
84 }
85
86
87
88
89
```

Camada de Serviços (Service)

- O Spring possui uma funcionalidade bastante interessante, que é a paginação.
- É bastante simples de implementar utilizando o Spring Data, que já disponibiliza duas interfaces prontas para serem usadas nesses casos: o **Page** e o **Pageable**.

Camada de Serviços (Service)

- O **Page** é uma coleção de objetos já paginados, que possui dados como o tamanho de uma página, quantos existem no total e qual a página que o usuário está visualizando.
- Já o **Pageable** recebe os parâmetros que o usuário pode usar para acessar uma página, como qual o tamanho e qual a página que deve ser acessada.
- A interface **JpaRepository** já possui um método **findAll** que recebe um objeto **Pageable** e faz a consulta fazendo paginação.
- Vamos implementar a consulta com paginação:

Camada de Serviços (Service)

```
8  import org.springframework.data.domain.Page;  
9  import org.springframework.data.domain.Pageable;
```

```
81  
82      public Page<UserDT0> getAllPage(Pageable page) {  
83          Page<User> users = userRepository.findAll(page);  
84          return users  
85              .map(UserDT0::convert);  
86      }  
87  
88  }  
89
```

Camada dos Controladores (Controllers)

- Os **controllers** mudam pouco em relação ao que já havíamos feito.
- Basicamente, eles chamarão a classe da camada de serviço.
- A classe continua com a mesma anotação **@RestController**, e os métodos com as anotações **@GetMapping**, **@PostMapping** e **@DeleteMapping**.
- Uma diferença é a injeção da dependência da classe de serviços **UserService**.

Camada dos Controladores (Controllers)

- Os métodos nela são os mesmos, com a diferença de que agora são chamados os métodos da camada de serviço em vez de manipular a lista em memória.
- Também vamos mudar o nome de algumas rotas para ficar mais próximo do que usaremos nos outros serviços; por exemplo:
 - **DELETE /removeUser/{cpf}** virou **DELETE /user/{id}** .

Camada dos Controladores (Controllers)

- Outro ponto importante nas rotas são os códigos de retorno.
- Por padrão, as rotas REST retornam o código 200, porém é uma boa prática mudar o tipo de retorno dependendo do tipo da rota.
- Nas rotas que criam um objeto, o melhor código de retorno é o 201, que indica que um recurso foi criado no servidor;
- Para uma rota DELETE, normalmente é retornado um código 204, indicando que a rota não terá nenhum dado de retorno.

Camada dos Controladores (Controllers)

```
src / main / java / com / montanha / user_api / controller / UserController.java / edit
1 package com.montanha.user_api.controller;
2
3 import java.util.List;
4
5 import org.bson.types.ObjectId;
6 import org.springframework.http.HttpStatus;
7 import org.springframework.web.bind.annotation.*;
8
9 import com.montanha.user_api.models.dto.UserDTO;
10 import com.montanha.user_api.services.UserService;
11
12 import jakarta.validation.Valid;
13 import lombok.RequiredArgsConstructor;
14
15 @RestController
16 @RequestMapping("/user")
17 @RequiredArgsConstructor
18 public class UserController {
19
```

```
18 public class UserController {
19
20     private final UserService userService;
21
22     @GetMapping
23     public List<UserDTO> getUsers() {
24         return userService.getAll();
25     }
26
27     @GetMapping("/{id}")
28     public UserDTO findById(@PathVariable String id) {
29         return userService.findById(id);
30     }
31
32     @PostMapping
33     @ResponseStatus(HttpStatus.CREATED)
34     public UserDTO newUser(@RequestBody @Valid UserDTO userDTO) {
35
36         return userService.save(userDTO);
37     }
38
39     @GetMapping("/{cpf}/cpf")
40     public UserDTO findByCpf(@PathVariable String cpf) {
41         return userService.findByCpf(cpf);
42     }
43
44     @DeleteMapping("/{id}")
45     @ResponseStatus(HttpStatus.NO_CONTENT)
46     public void delete(@PathVariable String id) {
47         userService.delete(id);
48     }
49
50     @GetMapping("/search")
51     public List<UserDTO> queryByName(
52         @RequestParam(name="nome", required = true) String nome) {
53         return userService.queryByName(nome);
54     }
55 }
```

Camada dos Controladores (Controllers)

- Uma novidade aqui é a rota **/user/search**, que fará a busca pelo nome recebido como parâmetro - o nome pode ser completo ou apenas parte do nome.
- Se o nome for completo, a rota retornará apenas um usuário;
- Se o usuário passar apenas parte do nome, pode ser retornada uma lista de usuários.

Camada dos Controladores (Controllers)

- Outra novidade é a anotação **@RequestParam**, que deve ser usada quando queremos passar parâmetros na URL para a rota.
- Veja que a anotação recebeu que o parâmetro é obrigatório.

Camada dos Controladores (Controllers)

- A chamada para essa rota pode ser feita pela URL <http://localhost:8080/user/search?nome=mar>
- Nesse caso, a resposta para a chamada será:

```
[
  {
    "nome": "marcela",
    "cpf": "123",
    "endereco": "Rua abc",
    "email": "marcela@email.com",
    "telefone": "1234-3454",
    "dataCadastro": "2019-11-17T21:04:51.701+0000",
  },
  {
    "nome": "marcelo",
    "cpf": "123",
    "endereco": "Rua abc",
    "email": "marcelo@email.com",
    "telefone": "1234-3454",
    "dataCadastro": "2019-11-17T21:04:51.701+0000",
  }
]
```


Camada dos Controladores (Controllers)

- Além da rota de **search**, também foram adicionadas outras duas rotas no **UserController**: as rotas de edição de usuário e a rota que retorna os usuários usando paginação.
- Agora vamos implementar a rota de edição de usuários.
- A rota de edição é parecida com a de criação de usuário.
- Ela recebe um JSON que representa um usuário e atualiza os dados do usuário conforme o que foi recebido.

Camada dos Controladores (Controllers)

```
54  
55     @PatchMapping("/{id}")  
56     public UserDTO editUser(@PathVariable String id,  
57                             @RequestBody UserDTO userDTO) {  
58         return userService.editUser(id, userDTO);  
59     }  
60 }
```

Camada dos Controladores (Controllers)

- Com essa implementação podemos atualizar o endereço do usuário que tem o CPF **123**, podemos chamar a rota PATCH <http://localhost:8080/user/123>, passando apenas os dados que queremos atualizar, como o endereço e o telefone:

```
{  
  "endereco": "Rua abc",  
  "telefone": "1234-3454"  
}
```

Camada dos Controladores (Controllers)

- Agora vamos implementar a busca de usuários usando paginação.
- A chamada é feita na rota GET <http://localhost:8080/user/pageable?size=2&page=1>.
- Note os parâmetros size e page.
- O size indica que as páginas terão tamanho 2, e que deve ser retornada a página 1 (lembrando que as páginas começam no índice 0).
- A listagem a seguir mostra o código dessa rota.

Camada dos Controladores (Controllers)

```
60  
61     @GetMapping("/pageable")  
62     public Page<UserDTO> getUsersPage(Pageable pageable) {  
63         return userService.getAllPage(pageable);  
64     }  
65 }  
66
```

Camada dos Controladores (Controllers)

```
1  {
2    "content": [
3      {
4        "id": "67325e5521d198418b5a9d41",
5        "nome": "João",
6        "cpf": "123",
7        "endereco": "Avenida Getulio Vargas",
8        "email": "joao@maria.com",
9        "telefone": "12345-9999",
10       "dataCadastro": "2024-11-11T16:43:17.503"
11     },
12     {
13       "id": "67325e6421d198418b5a9d42",
14       "nome": "João",
15       "cpf": "123",
16       "endereco": "Avenida Rondon",
17       "email": "joao@maria.com",
18       "telefone": "99999-9999",
19       "dataCadastro": "2024-11-11T16:43:32.572"
20     }
21   ],
22   "pageable": {
23     "pageNumber": 0,
24     "pageSize": 2,
25     "sort": {
26       "sorted": false,
27       "empty": true,
28       "unsorted": true
29     },
30     "offset": 0,
31     "paged": true,
32     "unpaged": false
33   },
34   "totalPages": 4,
35   "totalElements": 7,
36   "last": false,
37   "size": 2,
38   "number": 0,
```

Camada dos Controladores (Controllers)

- No retorno, existe um objeto chamado **content**, que possui a lista de usuários;
- Depois, existem diversos dados sobre paginação, como o número de página, o número total de elementos e o tamanho da página.
- Esses dados podem ser utilizados no front-end para a criação de uma tabela com paginação.
- Agora já temos uma versão funcional do user-api com integração com o banco de dados MongoDB.