

Tecnologias Para Back-End

Prof. JUNIO FIGUEIRÊDO

JUNIOINF@GMAIL.COM

AULA 08 – REQUISIÇÕES GET....

Spring

Paginação



Paginação de Médicos

- Agora vamos cuidar da **paginação** e da **ordenação**.
- Começaremos pela **paginação**.
- Queremos trazer por exemplo apenas 10 registros por página.
- Vamos configurar uma requisição para a API para passar de página quando necessário.
- Como paginação é algo comum, o Spring já tem um mecanismo para fazer isso. Em "**MedicoController.java**", no **método listar**, passaremos o **parâmetro Pageable**.
- Pageable é uma interface do próprio Spring.



- **Obs:** Cuidado na hora de importar.
- Selecione **Pageable** `org.springframework.data.domain`
- e não **Pageable** `java.awt.print`.
- A segunda não é aplicável ao Spring Framework

```
listar(Pageable paginacao)
```

Paginação de Médicos

- Daremos o nome **paginacao** ao parâmetro. Vamos passar o novo parâmetro dentro do método `.findAll`.

```
repository.findAll(paginacao)
```

- Com isso, o Spring montará a query automaticamente com o esquema de paginação.

Paginação de Médicos

- Substituiremos, também, o retorno do método.

```
List<DadosListagemMedico>
```

- Não será mais List, e sim Page.

```
Page<DadosListagemMedico>
```

- Ainda está retornando uma Lista de médico e também informações sobre a paginação
- Mais a frente vamos compreender melhor sobre esses dados da paginação.

Paginação de Médicos

- Vamos **alterar também o return**, que agora não precisará mais da chamada do **método .stream()**.
- Como o **findAll** devolve **um page** e **dentro d'ele já possui** um **método map()**, por esse motivo não vamos usar o método **.stream()**.

```
findAll(paginacao).stream().map(DadosListagemMedico::new).toList();
```

- **.toList()** também não será mais necessário, **pois ele devolve uma página a mesma já é uma lista**

```
return repository.findAll(paginacao).map(DadosListagemMedico::new);
```

```
@GetMapping
public Page<DadosListagemMedico> listar(Pageable paginacao){
    return repository.findAll(paginacao).map(DadosListagemMedico::new);
}
```


Paginação de Médicos

- Adaptamos o nosso método `Listar()`.
- Onde ele recebe um objeto pageable. Já o `findall` recebe a variável de referência `paginacao` “objeto” como parâmetro.

```
listar(Pageable paginacao){
```

```
return repository.findAll(paginacao).map(DadosListagemMedico::new);
```

- E ainda continua fazendo map. Só que é um map de (page de médicos).
- Que é **convertido em um page** de **dados listagem de médicos** “DTO”.
- Salve e inicie o servido.
- Vamos ao Insomnia.


Spring

GET ▼ http://localhost:8080/medicos Send ▼ 200 OK 803 ms 417 B

Parameters Body ▼ Auth ▼ Headers 1 Docs

Preview ▼ Headers 3 Cookies Timeline

Sem paginação



Enter a URL and send to get a response

Select a body type from above to send data in the body of a request


```
1 [
2   {
3     "nome": "Zel Ferreira",
4     "email": "rodrig.ferreira@voll.med",
5     "crm": "123458",
6     "especialidade": "ORTOPEDIA"
7   },
8   {
9     "nome": "Rodrigo Ferreira",
10    "email": "rodrigoss.ferreira@voll.med",
11    "crm": "123445",
12    "especialidade": "ORTOPEDIA"
13  },
14  {
15    "nome": "João Carlos",
16    "email": "joao.ferreira@voll.med",
17    "crm": "872934",
18    "especialidade": "ORTOPEDIA"
19  },
20  {
21    "nome": "Antonio Carlos",
22    "email": "joaeeeeo.ferreira@voll.med",
23    "crm": "871034",
24    "especialidade": "ORTOPEDIA"
25  }
26 ]
```

Spring

GET ▼ http://localhost:8080/medicos Send ▼ 200 OK 786 ms 731 B

Parameters Body ▼ Auth ▼ Headers 1 Docs Preview ▼ Headers 3 Cookies Timeline

Com paginação



Enter a URL and send to get a response

Select a body type from above to send data in the body of a request

```
1 {
2   "content": [
3     {
4       "nome": "Zel Ferreira",
5       "email": "rodrig.ferreira@voll.med",
6       "crm": "123458",
7       "especialidade": "ORTOPEDIA"
8     },
9     {
10      "nome": "Rodrigo Ferreira",
11      "email": "rodrigoss.ferreira@voll.med",
12      "crm": "123445",
13      "especialidade": "ORTOPEDIA"
14    },
15    {
16      "nome": "João Carlos",
17      "email": "joao.ferreira@voll.med",
18      "crm": "872934",
19      "especialidade": "ORTOPEDIA"
20    },
21    {
22      "nome": "Antonio Carlos",
23      "email": "joaeero.ferreira@voll.med",
24      "crm": "871034",
25      "especialidade": "ORTOPEDIA"
26    }
27  ]
28 }
```

Paginação de Médicos

- O JSON devolvido, agora, terá o atributo content, com o array da lista de médicos dentro dele.
- Ao final, encontraremos novas informações relacionadas à paginação.

```
19
"pageable": {
  "pageNumber": 0,
  "pageSize": 20,
  "sort": {
    "empty": true,
    "sorted": false,
    "unsorted": true
  },
  "offset": 0,
  "paged": true,
  "unpaged": false
},
```

```
},
"last": false,
"totalElements": 22,
"totalPages": 22,
"size": 1,
"number": 0,
"sort": {
  "empty": true,
  "sorted": false,
  "unsorted": true
},
"numberOfElements": 1,
"first": true,
"empty": false
}
```

Essas informações são devolvidas para o Front-End, para os dispositivos Mobile e etc...Para saber a quantidade de registro no BD e páginas, para desenhar o componente de paginação corretamente!!!

Paginação de Médicos

- Agora vamos controlar o número de registros exibidos.
- Para isso, passaremos, na URL, o parâmetro **?size**.
- Se **o igualarmos a 1**, teremos **a exibição de apenas um registro na tela**:

```
http://localhost:8080/medicos?size=1
```

Paginação de Médicos

Obs: Se não passarmos o parâmetro `size`, o Spring devolverá 20 registros por padrão.

- Para trazermos a página, vamos passar outro parâmetro na URL, após usar um `&`. Será o parâmetro `page`.
- Como a primeira página é representada por `page=0`, para trazer o próxima, traremos `page=1`. E assim sucessivamente.
- Com esse dois parâmetros, controlamos a paginação.

```
GET ▼ http://localhost:8080/medicos?size=1&page=3
```

Spring

Ordenação



Spring

- Agora vamos cuidar da ordenação.
- Para fazer isso, vamos remover os parâmetros `size` e `page`, adicionados por nós na aula anterior.
- Para mudar a ordenação, também usaremos um parâmetro na URL, chamado `sort`.
- Junto dele, `passamos` o `nome do atributo` na `entidade`.
- Se quisermos `ordenar pelo nome`, por exemplo, passaremos a URL `http://localhost:8080/medicos?sort=nome`.

Spring

- Se dispararmos a aquisição, veremos que a exibição dos registros será feita em ordem alfabética.
- Se quiser ordenar por outro parâmetro, basta substituir a informação depois de `sort`. `http://localhost8080/medicos?sort=crm`
- Por padrão, a ordenação acontece de maneira crescente.
- Mas é possível inverter isso, ordenando por ordem decrescente. Para isso, basta adicionar `desc` à URL.

```
http://localhost8080/medicos?sort=crm,desc
```

Spring

- É possível combinar com os parâmetro que vimos na aula de paginação.
- Basta adicioná-los na URL, sempre conectando-os com um **&**, como no exemplo abaixo:

```
http://localhost8080/medicos?sort=crm,desc&size=2&page=1
```

- Por padrão, o nome dos parâmetros é escrito em inglês.
- Porém, conseguimos customizar esses parâmetros no arquivo **"application.properties"**.

Spring

- Vamos voltar à IDE. O parâmetro `Pageable`, que usamos em `lista`, é opcional.

```
public Page<DadosListagemMedico> listar(Pageable paginacao){  
    return repository.findAll(paginacao).map(DadosListagemMedico::new);  
}
```

- Se voltarmos para o Insomnia e dispararmos a requisição sem nenhum parâmetro na URL, ela vai carregar todos os registros usando o padrão do Spring.

```
GET ▼ http://localhost:8080/medicos
```

Spring

- O padrão é 20 resultados por página, e na ordem em que cadastramos a informação no banco de dados.
- É possível, porém, alterar esse padrão.
- Em "MedicoController.java", podemos trocar o padrão da paginação adicionando uma anotação no parâmetro Pageable.
- O nome dela é @PageableDefault. Na sequência, abrimos parênteses e passamos os atributos size, page e sort.
- Podemos escolher o atributo que guiará a ordenação, passando entre chaves duplas.

Spring

- O padrão é 20 resultados por página, e na ordem em que cadastramos a informação no banco de dados.
- Por exemplo, se passarmos `lista(@PageableDefault(size = 10, sort = {"nome"}))`, isso significa que, caso não passemos parâmetros na URL, no Insomnia, o novo padrão será a exibição de 10 resultados por página, ordenados a partir do nome.
- Vamos salvar e, depois disso, podemos testar no Insomnia.

Spring

- Caso queiramos saber como a **query** está sendo feita no banco de dados, podemos configurar para que isso seja exibido para nós.
- A configuração é feita em "src > main > resources > **application.properties**".
- Lá, vamos adicionar **spring.jpa.show-sql=true**.
- Com isso, **os SQLs disparados no banco de dados** serão **impressos**.
- Com isso, conseguiremos ver as informações das queries na IDE após **fazermos a requisição** no Insomnia.

Spring

- Veja o log do IntelliJ.
- O parâmetro é difícil de ser visualizado, porque é exibido numa linha só.
- Para facilitar a visualização, vamos adicionar outro parâmetro, que passa as informações com quebras de linha.
- De volta a "application.properties", passaremos a propriedade `spring.jpa.properties.hibernate.format_sql=true`.
- Depois que salvarmos, veremos que tivemos sucesso.

- Conseguimos implementar nossa funcionalidade de listagem de médicos usando paginação, ordenação e usando o log para ver o que está sendo disparado.

Spring

