# 35. Paginação com parâmetros opcionais na requisição

Um requisição do tipo GET a um sistema com muitos registros pode gerar uma sobrecarga, para resolver esse problema faz-se a paginação. A paginação realiza a busca de registros com quantidades pré definidas, semelhante a páginas de produtos em sites de comércio eletronico.

## Em CategoriaResouce…

Vamos criar o EndPoint /categorias/page para isso precisamos de um método chamado findPage( ), devemos usar a seguinte anotação em seu cabeçalho:

```
@RequestMapping(value = "/pages", method = RequestMethod.DELETE)
```

FindPage( ) retorna uma ResponseEntity<?> implementada com Page<?> implementada com CategoriaDTO, ou seja, `ResponseEntity<Page<CategoriaDTO>> findPage( )`

Os parâmetros page, linesPerPage, orderBy e direction recebidos em findPage( ) são opcionais, e servem para configurar a página. Quando o cliente acessar /categorias/page ele poderá informar zero, um ou mais parâmetros para configurar a página, por isso cada parâmetro deve receber um valor padrão, a anotação @RequestParam com os atributos value e defaultValue serve para isso.

```
@RequestParam(value = "page", defaultValue = "0") Integer page
@RequestParam(value = "linesPerPage", defaultValue = "24") Integer linesPerPage
@RequestParam(value = "orderBy", defaultValue = "nome") String orderBy
@RequestParam(value = "direction", defaultValue = "ASC") String direction
```

Exemplo URI:
/categorias/page?page=0&linesPerPage=24&orderBy=nome

No corpo do método, a execução da linha a seguir

```
Page<Categoria> list = categoriaService.findPage(page, linesPerPage, orderBy, direction)
```

traz uma página(lista) de Categorias do banco de dados utilizando o método de serviço findPage( ). Como enviamos DTOs ao cliente é preciso converter essa lista de Categoria em uma lista de CategoriaDTO, por isso usamos

```
Page<CategoriaDTO> listDTO = list.map(obj -> new CategoriaDTO(obj))
```

e por fim retorna-se a ResponseEntity

```
return ResponseEntity.ok().body(listDTO);
```

CategoriaResource → findPage( )

```
@RequestMapping(value = "/page", method = RequestMethod.GET)
public ResponseEntity<Page<CategoriaDTO>> findPage(
        @RequestParam(value = "page", defaultValue = "0") Integer page,
        @RequestParam(value = "linesPerPage", defaultValue = "24") Integer linesPerPage,
        @RequestParam(value = "orderBy", defaultValue = "nome") String orderBy,
        @RequestParam(value = "direction", defaultValue = "ASC") String direction) {
    Page<Categoria> list = categoriaService.findPage(page, linesPerPage, orderBy, direction);
    Page<CategoriaDTO> listDTO = list.map(obj -> new CategoriaDTO(obj));
    return ResponseEntity.ok().body(listDTO);
}
```

## Em CategoriaService…

Criar um método chamado findPage( ) que retorna uma Page<Categoria> implementada com Categoria. Page<?> é uma classe do Spring Data que encapsula informações e operações sobre a paginação.

```
public Page<Categoria> findPage()
```

findPage( ) deve receber os seguintes parâmetros:

```
findPage(Integer page,  Integer linesPerPage, String orderBy, String direction)
```

`page` indica qual página requisitada (0,1, …, n), `linesPerPage` informa a quantidade de linhas por página (1, 2,…, n), `orderBy` indica o campo usado para ordenar a página (nome, preço) e `direction` da direção do ordenamento (ascendente, descendente).

Para retornar uma página é preciso criar outros objeto do tipo PageRequest e passar os mesmos parâmetros recebidos no método findPage( ) para ele:

```
PageRequest pageRequest =
new PageRequest(page, linesPerPage, Sort.Direction.valueOf(direction), orderBy);
```

O parâmetro direction ao ser passado para o objeto pageRequest deve ser convertido para o tipo Direction, isto é, Direction.valueOf(direction).

Por fim deve-se invocar o repositório de categoria passando o pageRequest com parâmetro:

```
return categoriaRepository.findAll(pageRequest);
```

CategoriaService → findPage( )

```
public Page<Categoria> findPage(Integer page,  Integer linesPerPage, String orderBy, String direction) {
    PageRequest pageRequest = new PageRequest(page, linesPerPage, Sort.Direction.valueOf(direction), orderBy);
    return categoriaRepository.findAll(pageRequest);
}
```

---

**Explicação @RequestParam:**

Podemos usar @RequestParam para extrair parâmetros de consulta, parâmetros de formular e também arquivos de uma requisição.

**Exemplo(1)** o nome do parâmetro é igual do nome da variável, as vezes podemos querer que estes nomes sejam diferentes (próximo exemplo).

```
@GetMapping("/api/foos")
@ResponseBody
public String getFoos(@RequestParam String id) {
    return "ID: " + id;
}
```

Este seria o retorno da requisição:

```
http://localhost:8080/api/foos?id=abc
----
ID: abc
```

**Exemplo(2)** parâmetro com nome diferente, ou seja, no método a variável é chamada de fooId porém na URI  será visto com o nome id.

```
@PostMapping("/api/foos")
@ResponseBody
public String addFoo(@RequestParam(name = "id") String fooId, @RequestParam String name) {
    return "ID: " + fooId + " Name: " + name;
}
```

**Exemplo(3)** é possível definir um valor padrão para um @RequestParam usando o atributo defaultValue. Isto permite que o usuário não precise informar todos os parâmetros quando acessa a URI.

```
    @GetMapping("/api/foos")
    @ResponseBody
    public String getFoos(@RequestParam(defaultValue = "test") String id) {
        return "ID: " + id;
    }
```

Fonte: https://www.baeldung.com/spring-request-param