

Desenvolvimento de APIs REST

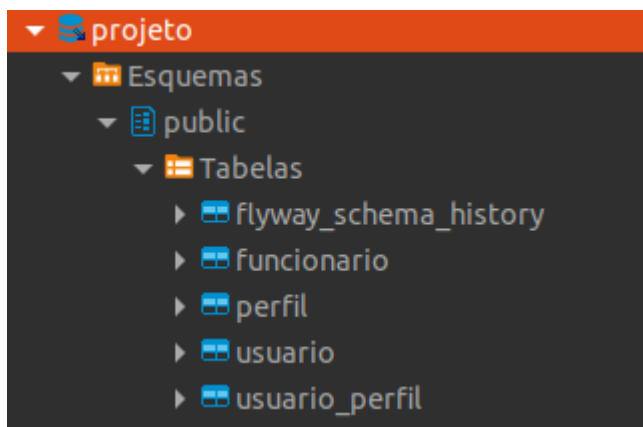
09 - JPQL, Paginação e Flyway

- Paginação
- Java Persistence Query Language
- Query Methods

Incluindo mais uma tabela



Utilizando o projeto **service-dto** vamos criar o arquivo `V03__criar_tabela_funcionario.sql` na pasta `db/migration`.



```
CREATE TABLE funcionario (  
  id_funcionario serial PRIMARY KEY,  
  nome varchar(60),  
  data_nascimento date,  
  salario NUMERIC  
);  
  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Carlos', '2000-05-10', 1000);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('João', '1999-06-11', 2000);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Arthur', '1998-02-11', 3550);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Marcos', '2001-02-22', 4200);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Adriana', '1998-01-22', 1000);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Carlos', '1974-11-21', 2500);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Carlos', '1966-08-12', 1950);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Yure', '1955-02-10', 10000);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Ana Beatriz', '1969-06-11', 2300);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Liliane', '1974-11-10', 3200);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Liliam', '2000-01-22', 6500);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Mariana', '2001-04-28', 2500);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Maria José', '1978-02-22', 2200);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Carlos André', '1976-12-22', 2300);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Carlos Arthur', '1988-08-12', 3800);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Carlos Rodrigues', '1989-02-12', 2800);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Lucas', '2001-05-10', 1000);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Roni', '2002-05-10', 1000);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Reinaldo', '2003-06-10', 1000);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Silvio', '2004-05-10', 1000);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Sergio', '2000-02-11', 1000);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Silas', '2000-01-10', 2000);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Juarez', '1988-05-10', 4000);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Flávio', '1978-05-10', 5000);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Augusto', '1977-06-10', 1000);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Rômulo', '1988-04-10', 1400);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Adriana', '2000-01-10', 1040);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Maiara', '2001-03-10', 1040);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Paulo', '1999-02-10', 1050);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Beatriz', '1976-01-10', 1500);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Laura', '1977-03-10', 1300);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('José Carlos', '1974-05-10', 2800);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Ronaldo', '1973-01-10', 4300);  
INSERT INTO funcionario (nome, data_nascimento, salario) VALUES ('Luis', '1974-04-13', 5250);
```

Criar Entidade, Repository e Controller



```
@Entity
public class Funcionario {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id_funcionario")
    private Long id;

    private String nome;
    private Double salario;

    @Column(name="data_nascimento")
    private LocalDate dataNascimento;
}
```

```
@RestController
@RequestMapping("/funcionarios")
public class FuncionarioController {
    @Autowired
    private FuncionarioRepository funcionarioRepository;

    @GetMapping
    public ResponseEntity<List<Funcionario>> listar(){
        List<Funcionario> funcionarios =
            funcionarioRepository.findAll();
        return ResponseEntity.ok(funcionarios);
    }
}
```

```
@Repository
public interface FuncionarioRepository extends JpaRepository<Funcionario, Long> {
}
```

The screenshot shows a REST client interface with a GET request to `http://localhost:8080/funcionarios`. The response is a 200 OK status with a 26 ms response time and 2.62 KB of data. The response body is displayed in JSON format, showing a list of three employees:

```
[
  {
    "id": 1,
    "nome": "Carlos",
    "salario": 1000.0,
    "dataNascimento": "2000-05-10"
  },
  {
    "id": 2,
    "nome": "João",
    "salario": 2000.0,
    "dataNascimento": "1999-06-11"
  },
  {
    "id": 3,
    "nome": "Arthur",
    "salario": 3000.0,
    "dataNascimento": "1998-07-12"
  }
]
```

Paginação



A paginação ajuda na performance de uma API, pode ser utilizada quando é feita uma requisição de consulta, serve para filtrar a quantidade de registros que você quer buscar. Exemplo, se você estiver realizando uma consulta em uma base de dados, além dos filtros da sua query, você também pode restringir a quantidade de registros que deseja retornar em um intervalo de páginas.

Para usar a paginação devemos utilizar a classe Page

```
@RestController
@RequestMapping("/funcionarios")
public class FuncionarioController {
    @Autowired
    private FuncionarioRepository funcionarioRepository;

    @GetMapping
    public ResponseEntity<List<Funcionario>> listar() {
        List<Funcionario> funcionarios = funcionarioRepository.findAll();
        return ResponseEntity.ok(funcionarios);
    }

    @GetMapping("/pagina")
    public ResponseEntity<Page<Funcionario>> listar(Pageable pageable) {
        Page<Funcionario> funcionarios = funcionarioRepository.findAll(pageable);
        return ResponseEntity.ok(funcionarios);
    }
}
```

Paginação



Testando no Postman são exibidas as informações sobre a paginação.

O objeto retornado possui alguns atributos específicos:

- Content: lista de dados retornados (apenas a primeira página)
- Pageable: contém informações sobre a paginação
 - sort - ordenação
 - offset - número do registro inicial da página
 - pageSize - total de registros por página
- totalPages: total de páginas
- totalElements: total de registros
- last: se é a última página

```
1 {
2   "content": [
3     {
4       "id": 1,
5       "nome": "Carlos",
6       "salario": 1000.0,
7       "dataNascimento": "2000-05-10"
8     },
9     {
10      "id": 2,
11      "nome": "João",
12      "salario": 2000.0,
13      "dataNascimento": "1999-06-11"
14    }
15  ]
16 }
```

```
118   "nome": "Silvio",
119   "salario": 1000.0,
120   "dataNascimento": "2004-05-10"
121 }
122 ],
123 "pageable": {
124   "sort": {
125     "empty": true,
126     "sorted": false,
127     "unsorted": true
128   },
129   "offset": 0,
130   "pageNumber": 0,
131   "pageSize": 20,
132   "paged": true,
133   "unpaged": false
134 },
135 "totalPages": 2,
136 "totalElements": 34,
137 "last": false,
```

Paginação



Valores Default

Podemos passar valores default caso os parâmetros da paginação não sejam informados, utilizando a anotação **@PageableDefault**. No **@PageableDefault** podemos definir atributos como **page** onde define em qual página que será retornada, o **size** que retorna o total de elementos, o **sort** para ordenação dos registros pelo nome do campo e o **direction** que define o tipo de ordenação ascendente ou descendente.

```
@RestController
@RequestMapping("/funcionarios")
public class FuncionarioController {

    @Autowired
    private FuncionarioRepository funcionarioRepository;

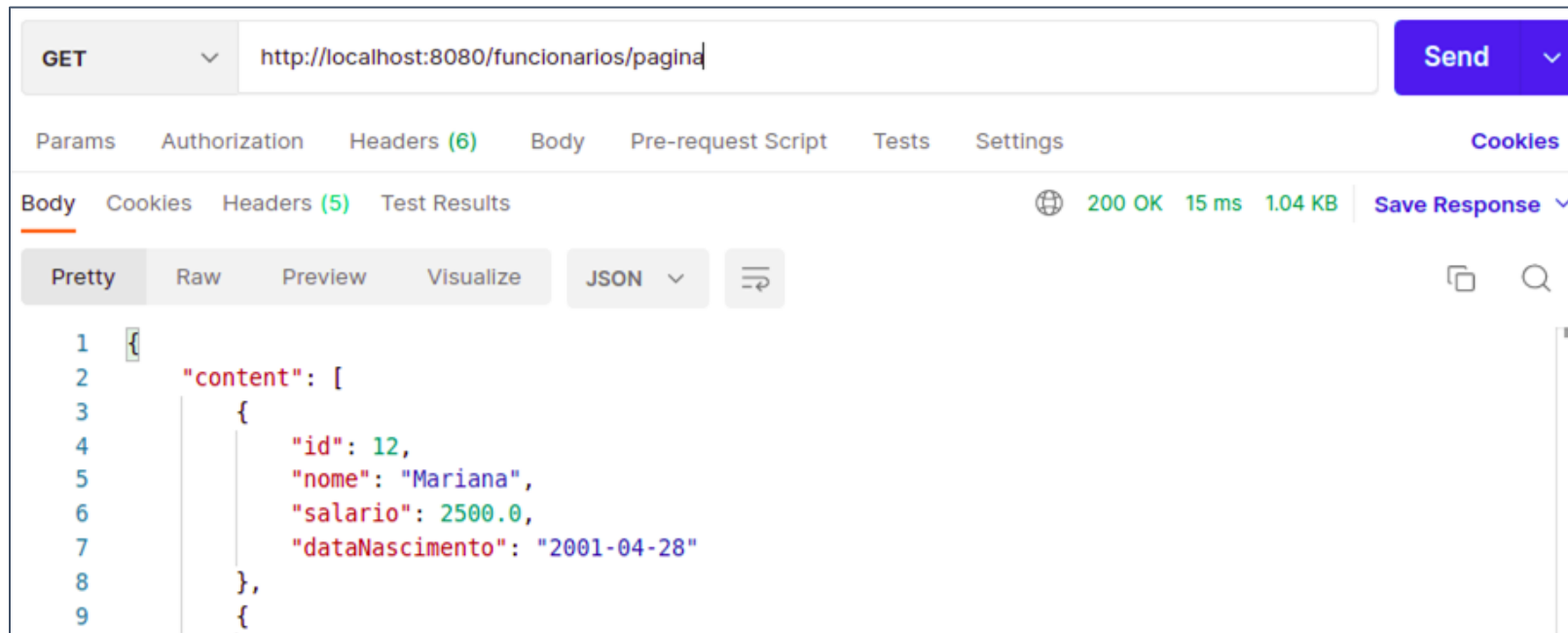
    @GetMapping
    public ResponseEntity<List<Funcionario>> listar() {
        List<Funcionario> funcionarios = funcionarioRepository.findAll();
        return ResponseEntity.ok(funcionarios);
    }

    @GetMapping("/pagina")
    public ResponseEntity<Page<Funcionario>> listar(
        @PageableDefault(sort="nome", direction = Sort.Direction.ASC,
            page = 3, size = 8) Pageable pageable) {
        Page<Funcionario> funcionarios = funcionarioRepository.findAll(pageable);
        return ResponseEntity.ok(funcionarios);
    }
}
```

Paginação



Exibindo no Postman



```
46     "id": 22,
47     "nome": "Silas",
48     "salario": 2000.0,
49     "dataNascimento": "2000-01-10"
50   }
51 },
52   "pageable": {
53     "sort": {
54       "empty": false,
55       "sorted": true,
56       "unsorted": false
57     },
58     "offset": 24,
59     "pageNumber": 3,
60     "pageSize": 8,
61     "paged": true,
62     "unpaged": false
63   },
64   "totalPages": 5,
65   "totalElements": 34,
66   "last": false,
67   "number": 3,
68   "sort": {
```

Paginação



Podemos alterar os da página como passando o parâmetro na url no Postman:

`http://localhost:8080/funcionarios/pagina?page=0&size=2`

Veremos as alterações no objeto retornado:

- **content:** apenas dois itens
- **pageable.offset:** página 0
- **pageable.pageSize:** 2
- **totalPages:**17

```
1 {
2   "content": [
3     {
4       "id": 1,
5       "nome": "Carlos",
6       "salario": 1000.0,
7       "dataNascimento": "2000-05-10"
8     },
9     {
10      "id": 2,
11      "nome": "João",
12      "salario": 2000.0,
13      "dataNascimento": "1999-06-11"
14    }
15  ],
16  "pageable": {
17    "sort": {
18      "empty": true,
19      "sorted": false,
20      "unsorted": true
21    },
22    "offset": 0,
23    "pageNumber": 0,
24    "pageSize": 2,
25    "paged": true,
26    "unpaged": false
27  },
28  "totalPages": 17,
29  "totalElements": 34,
```


Paginação



Página 0 com 10 elementos ordenada por data de nascimento e nome

**`http://localhost:8080/funcionarios/pagina?page=0&size=10
&sort=dataNascimento,nome,asc`**

GET http://localhost:8080/funcionarios/pagina?page=0&size=10&sort=dataNascimento,nome,asc

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Body Cookies Headers (5) Test Results 200 OK 18 ms 1.2 KB

Pretty Raw Preview Visualize JSON

```
1 {
2   "content": [
3     {
4       "id": 8,
5       "nome": "Yure",
6       "salario": 10000.0,
7       "dataNascimento": "1955-02-10"
8     },
9     {
10      "id": 7,
11      "nome": "Carlos",
12      "salario": 1950.0,
13      "dataNascimento": "1966-08-12"
14    },
15    {
16      "id": 9,
17      "nome": "Ana Beatriz",
18      "salario": 2300.0,
19      "dataNascimento": "1969-06-11"
20    },
21    {
22      "id": 33,
23      "nome": "Ronaldo",
24      "salario": 4300.0,
25      "dataNascimento": "1973-01-10"
26    }
27  ]
28 }
```

Linguagem de consultas Objeto Relacional



JPQL(Java Persistence Query Language)

É uma linguagem de consulta ORM em que usamos classes e objetos, diferente do SQL que trabalha com tabelas. É também uma especificação da JPA, e apesar de trabalhar com classes é semelhante a SQL. Os nomes das classes Java e dos campos de dados são case sensitive.

Uma query é formada pelas cláusulas **SELECT** e **FROM** e mais quatro cláusulas opcionais com o seguinte formato:

SELECT ... FROM ... [WHERE ...] [GROUP BY ... [HAVING ...]] [ORDER BY ...]

A estrutura das queries **DELETE** e **UPDATE** é no seguinte formato:

DELETE FROM ... [WHERE ...]
UPDATE ... SET ... [WHERE ...]

Linguagem de consultas Objeto Relacional



Exemplo:

Na interface **FuncionarioRepository** definimos uma nova assinatura de método para fazer o filtro para mostrar os salários entre os valores passados nos parâmetros **valorMinimo** e **valorMaximo**

```
@Repository
public interface FuncionarioRepository extends JpaRepository<Funcionario, Long> {
    @Query("SELECT f FROM Funcionario f where f.salario >= :valorMinimo AND f.salario <= :valorMaximo")
    Page<Funcionario> buscarSalario(Double valorMinimo, Double valorMaximo, Pageable pageable);
}
```

No controller **FuncionarioController** incluir o método abaixo

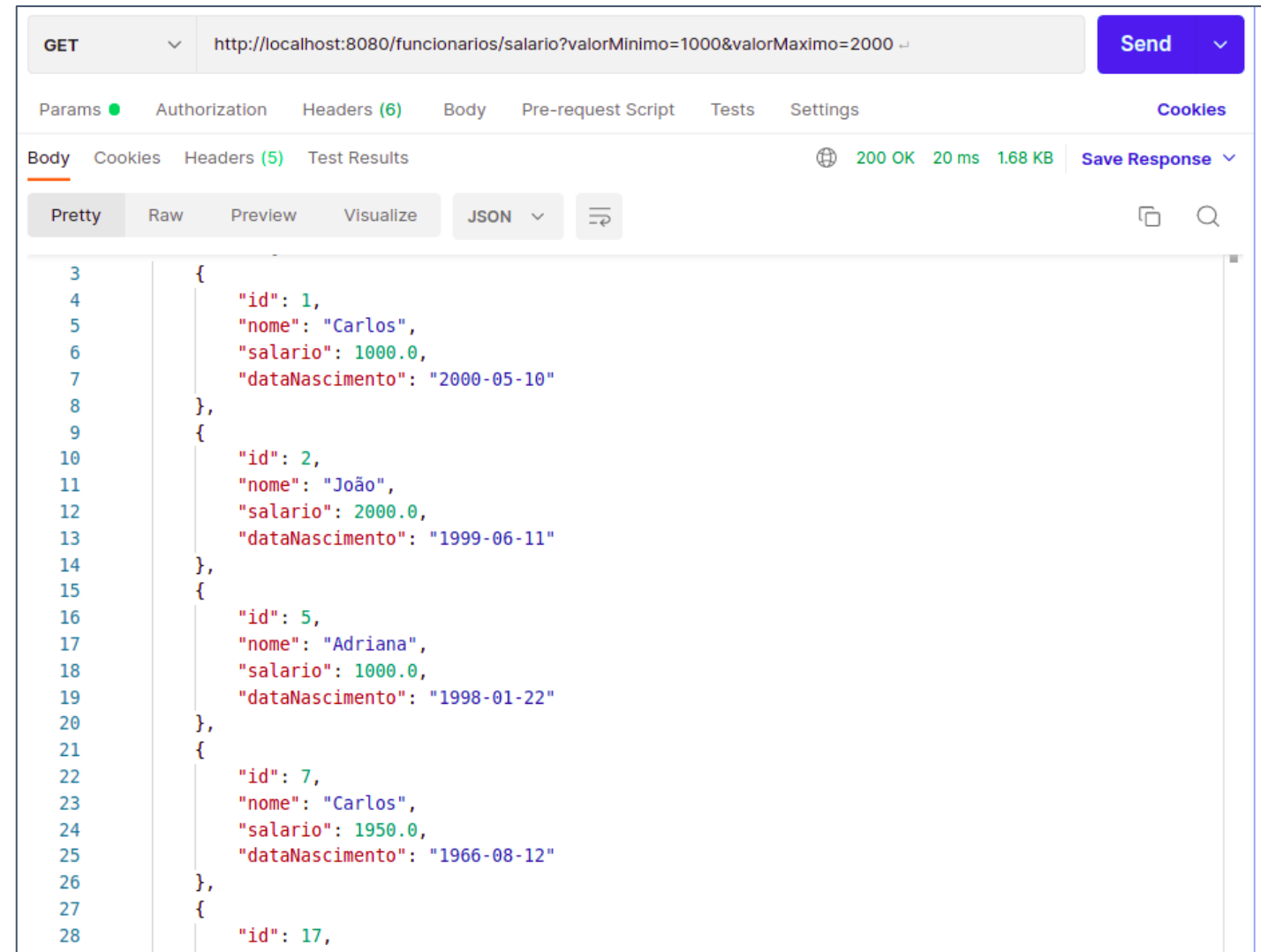
```
@GetMapping("/salario")
public ResponseEntity<Page<Funcionario>> listarSalarios(@RequestParam Double valorMinimo, @RequestParam Double valorMaximo, Pageable pageable) {
    Page<Funcionario> funcionarios = funcionarioRepository.buscarSalario(valorMinimo, valorMaximo, pageable);
    return ResponseEntity.ok(funcionarios);
}
```

Linguagem de consultas Objeto Relacional



Testar no Postman

`http://localhost:8080/funcionarios/salario?valorMinimo=1000
&valorMaximo=2000`



Linguagem de consultas Objeto Relacional



Se o usuário não passar os parâmetros será retornado o erro 400 Bad Request

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:8080/funcionarios/salarid
- Response Status:** 400 Bad Request
- Response Time:** 48 ms
- Response Size:** 5.47 KB
- Response Body (JSON):**

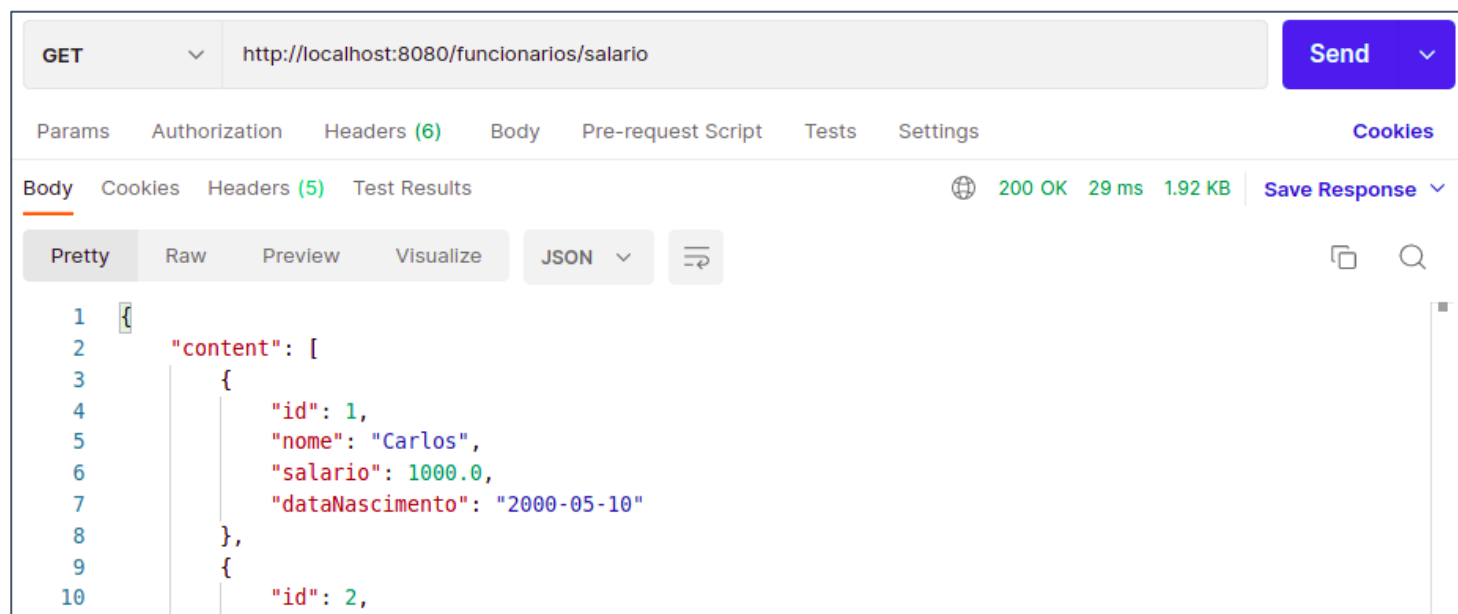
```
{
  "timestamp": "2022-09-06T15:55:06.198+00:00",
  "status": 400,
  "error": "Bad Request",
  "trace": "org.springframework.web.bind.MissingServletRequestParameterException: Required request parameter 'valorMinimo' for method parameter type Double is not present",
  "message": "Required request parameter 'valorMinimo' for method parameter type Double is not present",
  "path": "/funcionarios/salario"
}
```

Linguagem de consultas Objeto Relacional



Podemos inserir um valor default no RequestParam para evitar este erro. No exemplo abaixo valorMinimo valor padrão 0 e valor máximo 20000

```
@GetMapping("/salario")
public ResponseEntity<Page<Funcionario>> listarSalarios(
    @RequestParam(defaultValue = "0") Double valorMinimo,
    @RequestParam(defaultValue = "20000") Double valorMaximo,
    Pageable pageable) {
    Page<Funcionario> funcionarios = funcionarioRepository.buscarSalario(valorMinimo, valorMaximo, pageable);
    return ResponseEntity.ok(funcionarios);
}
```



Linguagem de consultas Objeto Relacional



Podemos também definir os parâmetros de paginação junto com os parâmetros da requisição

`http://localhost:8080/funcionarios/salario?valorMinimo=1000&valorMaximo=5000&size=2&page=2`

The screenshot shows a REST client interface with a GET request to `http://localhost:8080/funcionarios/salario?valorMinimo=1000&valorMaximo=5000&size=2&page=2`. The response is a JSON object with a status of 200 OK, 22 ms, and 626 B. The response body is displayed in JSON format, showing a list of two employees and pagination details.

```
1 {
2   "content": [
3     {
4       "id": 5,
5       "nome": "Adriana",
6       "salario": 1000.0,
7       "dataNascimento": "1998-01-22"
8     },
9     {
10      "id": 6,
11      "nome": "Carlos",
12      "salario": 2500.0,
13      "dataNascimento": "1974-11-21"
14    }
15  ],
16  "pageable": {
17    "sort": {
18      "empty": true,
19      "sorted": false,
20      "unsorted": true
21    },
22    "offset": 4,
23    "pageNumber": 2,
```

Linguagem de consultas Objeto Relacional



Adicionando uma nova assinatura de método com a query em JPQL para buscar parte do nome na interface [FuncionarioRepository](#)

```
@Query("SELECT f FROM Funcionario f WHERE UPPER(f.nome) like UPPER(CONCAT('%', :paramNome, '%'))")
Page<Funcionario> buscarPorNome(String paramNome, Pageable pageable);
```

Inserir o método [buscarPorNome](#) na classe [FuncionarioController](#)

```
@GetMapping("/nome")
public ResponseEntity<Page<Funcionario>> buscarPorNome(@RequestParam(defaultValue = "") String paramNome, Pageable pageable) {
    Page<Funcionario> funcionarios = funcionarioRepository.buscarPorNome(paramNome, pageable);
    return ResponseEntity.ok(funcionarios);
}
```


Linguagem de consultas Objeto Relacional



Testando no Postman passando o parâmetro

http://localhost:8080/funcionarios/nome?
paramNome=Ana

The screenshot shows the Postman interface with a GET request to `http://localhost:8080/funcionarios/nome?paramNome=Ana`. The response is a JSON array of employee objects. The status is 200 OK, and the response time is 10 ms. The response body is displayed in the 'Body' tab, showing a JSON array of employee objects.

```
1 {
2   "content": [
3     {
4       "id": 5,
5       "nome": "Adriana",
6       "salario": 1000.0,
7       "dataNascimento": "1998-01-22"
8     },
9     {
10      "id": 9,
11      "nome": "Ana Beatriz",
12      "salario": 2300.0,
13      "dataNascimento": "1969-06-11"
14    },
15    {
16      "id": 12,
17      "nome": "Mariana",
18      "salario": 2500.0,
19      "dataNascimento": "2001-04-28"
20    },
21    {
22      "id": 27,
23      "nome": "Adriana",
24      "salario": 1040.0,
25      "dataNascimento": "2000-01-10"
26    }
27  ],
28  "pageable": {
```

Consultas nativa + Interface DTO



Em alguns casos, é necessário realizar consultas complexas no banco de dados, que não são suportadas pela JPQL. Nestes casos é possível utilizar a anotação `@Query` com o atributo `nativeQuery`, indicando que é uma consulta nativa do banco de dados utilizado e não uma consulta jpql.

Nestes casos é possível criar uma interface DTO que tenha a “assinatura” de métodos compatível com as colunas retornadas pela consulta.

```
public interface FuncionarioSalarioDTO {  
    public Integer getIdade();  
    public Double getMediaSalario();  
    public Double getMenorSalario();  
    public Double getMaiorSalario();  
    public Double getTotalFuncionarios();  
}
```

```
@Query(value = "select date_part('year',age(now(), data_nascimento)) as idade, "+  
               "      avg(salario) as mediaSalario, " +  
               "      min(salario) as menorSalario, " +  
               "      max(salario) as maiorSalario, " +  
               "      count(*) as totalFuncionarios " +  
               " from funcionario " +  
               " group by idade " +  
               " having count(*)>1 "+  
               " order by idade desc ",nativeQuery = true)  
List<FuncionarioSalarioDTO> buscaSalariosPorIdade();
```

Consultas nativa + Interface DTO



```
@GetMapping("/salarios-por-idade")
public ResponseEntity<List<FuncionarioSalarioDTO>> buscaSalariosPorIdade() {
    return ResponseEntity.ok(funcionarioRepository.buscaSalariosPorIdade());
}
```

GET http://localhost:8080/funcionarios/salarios-por-idade

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (14) Test Results 200 OK 37 ms 1.35 KB

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "idade": 48,
4     "mediaSalario": 4705.0,
5     "menorSalario": 2800.0,
6     "maiorSalario": 5340.0,
7     "totalFuncionarios": 4.0
8   },
9   {
10    "idade": 47,
11    "mediaSalario": 2850.0,
12    "menorSalario": 2500.0,
13    "maiorSalario": 3200.0,
14    "totalFuncionarios": 2.0
15  },
16  {
17    "idade": 45,
18    "mediaSalario": 1533.3333333333333,
19    "menorSalario": 1000.0,
20    "maiorSalario": 2300.0,
21    "totalFuncionarios": 3.0
22  },
23 ]
```

“Query Methods”



Podemos transformar o nome dos atributos em métodos que são implementados pelo Spring Data JPA.
Basta usar o padrão **findBy<nome do atributo>**

Adicionando as linhas abaixo em destaque teremos o mesmo resultado do método **buscarSalario** e **buscarPorNome**

```
Page<Funcionario> findBySalarioBetween(Double valorMinimo, Double valorMaximo, Pageable pageable);  
Page<Funcionario> findByNomeContainingIgnoreCase(String paramNome, Pageable pageable);
```

O Spring data permite criar consultas personalizadas baseada nos nomes dos métodos do Repositório:

List<Usuario> findByNomels(String parametro) List<Usuario> findByNomeEquals(String parametro)	Select * from usuario where nome="Luis"	List<Usuario> findOrderByldadeDescTop3()	order by idade desc limit 3
List<Usuario> findByNomelsNot(String parametro)	where not nome="Luis"	List<Usuario> findByNomeldade(String parametro, Integer valor)	nome="parametro" and idade=valor
List<Usuario> findByNomelsNull()	where nome is null	List<Usuario> findByldadeBetween(Integer inicio, Integer fim) List<Usuario> findByldadeIn(Collection<Integer> valores)	idade between inicio and fim idade in (15, 29, 32, 56)
List<Usuario> findByNomeStartingWith(String prefixo) List<Usuario> findByNomeContaining(String texto) List<Usuario> findByNameLike(String padraoLike)	nome like "A%" nome like "%A%" nome like <padraoLike>	List<Usuario> findByldadeLessThan(Integer valor) List<Usuario> findByldadeLessThanEqual(Integer valor) List<Usuario> findByldadeGreaterThan(Integer valor) List<Usuario> findByldadeGreaterThanEqual(Integer valor)	idade < valor idade <= valor idade > valor idade >= valor

Mais exemplos e informações em <https://www.baeldung.com/spring-data-derived-queries>

“Query Methods”



Alterar no **FuncionarioController** para utilizar os novos métodos

GET ⌵ http://localhost:8080/funcionarios/nome?paramNome=jo Send ⌵

Params ● Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ⌵ ≡

```
1 {
2   "content": [
3     {
4       "id": 2,
5       "nome": "João",
6       "salario": 2000.0,
7       "dataNascimento": "1999-06-11"
8     },
9     {
10      "id": 13,
11      "nome": "Maria José",
12      "salario": 2200.0,
13      "dataNascimento": "1978-02-22"
14    },
15    {
16      "id": 32,
```

```
@GetMapping("/salario")
public ResponseEntity<Page<Funcionario>> listarSalarios(
    @RequestParam(defaultValue = "0") Double valorMinimo,
    @RequestParam(defaultValue = "20000") Double valorMaximo,
    Pageable pageable) {
    Page<Funcionario> funcionarios =
        funcionarioRepository.findBySalarioBetween(valorMinimo, valorMaximo, pageable);
    return ResponseEntity.ok(funcionarios);
}

@GetMapping("/nome")
public ResponseEntity<Page<Funcionario>> buscarPorNome(
    @RequestParam(defaultValue = "") String paramNome,
    Pageable pageable) {
    Page<Funcionario> funcionarios =
        funcionarioRepository.findByNomeContainingIgnoreCase(paramNome, pageable);
    return ResponseEntity.ok(funcionarios);
}
```