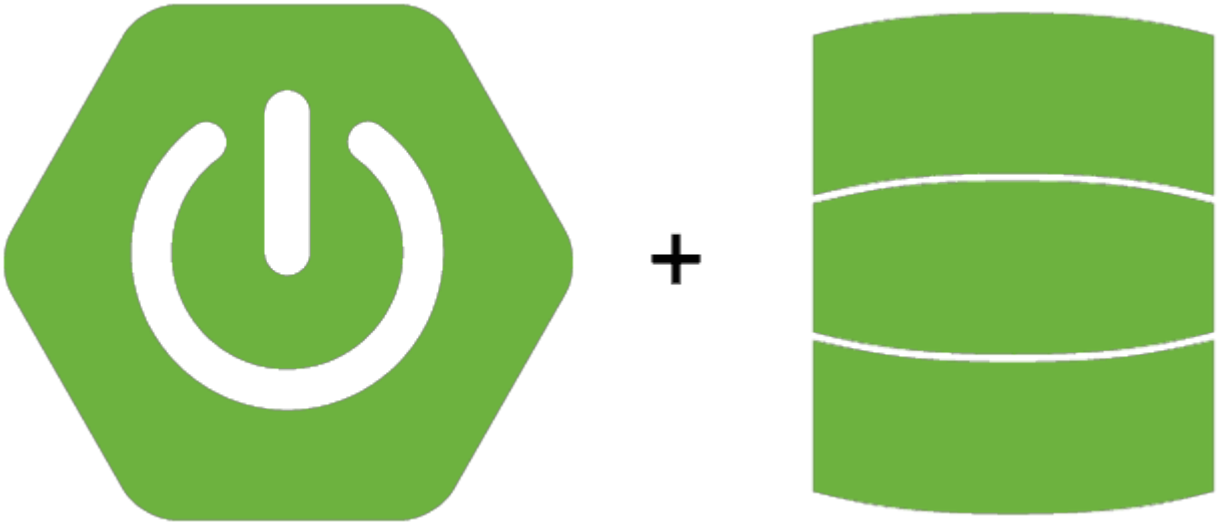
Fabiano Góes (e-Programar)

Follow 120 Followers About

API Rest com paginação usando Spring Data e @Query

 Fabiano Góes (e-Programar) Nov 19, 2018 · 4 min read



1. O Problema

Neste artigo vamos resolver o seguinte problema:

Precisamos de uma API Rest que me exponha uma consulta de Clientes.

- Deve permitir **filtro** por nome **OU** email.
- A **lista** retornada deve estar **paginada**.

recursos já testado pelo pessoal do Spring =).

Vamos usar:
- **Spring Data** para para fazer nosso acesso aos dados e facilitar a construção de nossas queries com paginação e filtro.
- **Spring Rest** Controller para expor nossa **API**.
- **Lombok** para automatizar nossos **gettters**, **setters** e **builder**.

2. A Solução

Primeiro vamos criar nosso modelo de Cliente:

```
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Entity
public class Customer {

    @Id @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;
    private String email;

}
```

Perceba que temos algumas Annotaions do **Lombok**:

@Data: Automatiza nossos getters e setter.

@NoArgsConstructor: Cria um construtor vazio.
além da **@Entity** que é do nosso conhecido JPA.

Com nosso modelo definido, vamos criar nossa Repository para implementar a solução de Filtro e paginação na camada de acesso aos dados.

```
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

@Repository
public interface CustomerRepository extends JpaRepository<Customer,
Long> {

    @Query("FROM Customer c " +
           "WHERE LOWER(c.name) like %:searchTerm% " +
           "OR LOWER(c.email) like %:searchTerm%")
    Page<Customer> search(
            @Param("searchTerm") String searchTerm,
            Pageable pageable);

}
```

Aqui usamos o **@Query** para customizar nosso filtro pelo nome OU email.
usamos o retorno como **Page<Customer>** porque o **Spring** já faz a mágica para nós e implementa a paginação =) e como parâmetro adicionamos um Pageable para passar ao Spring como queremos paginar:
- qual a pagina que queremos.
- a quantidade de registros por pagina.
- e a ordenação.

Ok, como nossa camada de acesso a dados implementada vamos implementar nossa Service.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageImpl;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort;
```

```
public class CustomerService {

    @Autowired
    CustomerRepository repository;

    public Page<Customer> search(
            String searchTerm,
            int page,
            int size) {
        PageRequest pageRequest = PageRequest.of(
                page,
                size,
                Sort.Direction.ASC,
                "name");

        return repository.search(
                searchTerm.toLowerCase(),
                pageRequest);
    }

    public Page<Customer> findAll() {
        int page = 0;
        int size = 10;
        PageRequest pageRequest = PageRequest.of(
                page,
                size,
                Sort.Direction.ASC,
                "name");
        return new PageImpl<>(
                repository.findAll(),
                pageRequest, size);
    }

}
```

Aqui usamos um **PageRequest** para passar nosso parâmetro **Pageable** que é como o Spring saberá os detalhes da paginação.
Também usei um **findAll()** pra demostrar como transformar uma List<Customer> em Page<Customer> usando new **PageImpl**.

Legal, agora só falta expor nosso negócio através de uma API Rest.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
```

```
public class CustomerController {

    @Autowired
    CustomerService service;

    @GetMapping("/search")
    public Page<Customer> search(
            @RequestParam("searchTerm") String searchTerm,
            @RequestParam(
                    value = "page",
                    required = false,
                    defaultValue = "0") int page,
            @RequestParam(
                    value = "size",
                    required = false,
                    defaultValue = "10") int size) {
        return service.search(searchTerm, page, size);

    }

    @GetMapping
    public Page<Customer> getAll() {
        return service.findAll();
    }

}
```

Aqui cabe apenas comentar que recebemos os parâmetros: **page** e **size** como **required** = false e definimos um valor default para eles.
Um detalhe importante é que a primeira pagina **começa do zero** não do 1.

agora pode testar através da url:
http://localhost:8080/customers/search?searchTerm=james


```
{
    "content": [
        {
            "id": 1,
            "name": "James Gosling",
            "email": "jamesgosling@java.com"
        }
    ],
    "pageable": {
        "sort": {
            "sorted": true,
            "unsorted": false,
            "empty": false
        },
        "offset": 0,
        "pageSize": 10,
        "pageNumber": 0,
        "paged": true,
```

```
    "last": true,
    "size": 10,
    "number": 0,
    "first": true,
    "numberOfElements": 1,
    "sort": {
        "sorted": true,
        "unsorted": false,
        "empty": false
    },
    "empty": false
}
```

e também podemos testar o findAll() para ver a ordenação:

```
{
    "content": [
        {
            "id": 1,
            "name": "James Gosling",
            "email": "jamesgosling@java.com"
        },
        {
            "id": 2,
            "name": "Guido van Rossum",
            "email": "guidovanrossum@python.com"
        },
        {
            "id": 3,
            "name": "Larry Wall",
            "email": "larrywall@pearl.com"
        },
        {
            "id": 4,
            "name": "Linus Torvalds",
            "email": "linustorvalds@linux.com"
        },
        {
            "id": 5,
            "name": "Rod Johnson",
            "email": "Rod Johnson@springframework.com"
        }
    ],
    "pageable": {
        "sort": {
            "sorted": true,
            "unsorted": false,
            "empty": false
        },
        "offset": 0,
        "pageSize": 10,
        "pageNumber": 0,
        "paged": true,
        "unpaged": false
    },
    "totalPages": 1
```

> *Mas neste caso a melhor opção quando vai precisar de paginação é já fazer o extends de "**PagingAndSortingRepository**" assim o findAll() já vem paginado =).*

3. Conclusão

Conseguimos resolver nosso problema utilizando bastante coisa pronta do eco sistema do **Spring**, não precisamos nos preocupar em nenhuma lógica de como criar uma paginação apenas customizamos nossa **Query** pra fazer o filtro usando like por um ou outro campo.
Além de usar o poder do **Spring Dat**a com **JPA** e o Spring Rest Controller para expor nosso negócio através de uma API Rest.
Também utilizamos um biblioteca interessante que é o Lombok, assim não precisamos fazer aquele trabalho repetitivo e entediante de criar os getters, setters, construtores e builder.
Espero que gostem da dica de hoje.

O código completo pode ser visto no **Github**:
https://github.com/fabianogoes/demo-pageable.git

Referencias:

- https://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html
- https://docs.spring.io/spring-data/rest/docs/2.0.0.M1/reference/html/paging-chapter.html
- https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.at-query