



Introdução a Web Services REST com Spring Framework

*Parte 8 → HATEOAS: Hypermedia As The
Engine of Application State*

Prof. Me. Jorge Luís Gregório

www.jlgregorio.com.br



Agenda

- **O que é HATEOAS?**
 - Maneiras de prover HATEOAS
 - Padrões IANA
- **PRÁTICA 01**
 - Adicionando suporte a HATEOAS na aplicação
- **PRÁTICA 02**
 - Paginação de resultados
 - Paginação HATEOAS

A large, stylized green leaf graphic is positioned on the left side of the slide, partially overlapping the central text box. The leaf is light green with a darker green outline and a central vein.

O que é HATEOAS?

HATEOAS - Hypermedia As The Engine of Application State

- É uma das restrições arquiteturais definidas por Roy Thomas Fielding, criador do REST;
- As respostas devem incluir *links* que permitem navegar entre os *endpoints* da API de serviços, provendo uma navegação dinâmica entre os dados da aplicação, assim como em uma página HTML;

```
HTTP/1.1 200 OK
Content-Type: application/vnd.acme.account+json
Content-Length: ...

{
  "account": {
    "account_number": 12345,
    "balance": {
      "currency": "usd",
      "value": 100.00
    },
    "links": {
      "deposit": "/accounts/12345/deposit",
      "withdraw": "/accounts/12345/withdraw",
      "transfer": "/accounts/12345/transfer",
      "close": "/accounts/12345/close"
    }
  }
}
```

Duas maneiras de prover HATEOAS

- [RFC 5988](#) (*web linking*)
 - Define uma estrutura para construir links que definem as relações entre os recursos da API de serviços ou externos;
 - Cada link contém as seguintes propriedades:
 - **Target URI:** representado pelo atributo href
 - **Relation Type:** define como o recurso está relacionado ao link. É representado pelo atributo rel.
 - **Atributos:** atributos adicionais do link, que incluem hreflang, media, title e type e outros parâmetros.
- JSON Hypermedia API Language (HAL)
 - Define convenções para expressar controles *hypermedia*, como *links* para recursos do serviço ou externos.
 - A proposta HAL ainda está em desenvolvimento, mas a ideia é padronizar as respostas HATEOAS para facilitar a comunicação entre serviços e desenvolvedores;
 - Há dois tipos associados ao HAL:
 - media type: application/hal+xml;
 - media type: application/hal+json;

A estrutura do HAL

- Cada link possui as três seguintes propriedades:
 - **Target URI:** indica o destino do *link*, representado pelo atributo **href**;
 - **Link Relation:** define como o recurso está relacionado ao destino. É definido pelo atributo **rel**;
 - **Type:** define o *media type* da resposta, representado pelo atributo **type**;

Embedded resource

```
{
  "_links": {
    "self": {
      "href": "http://example.com/api/book/hal-cookbook"
    }
  },
  "_embedded": {
    "author": {
      "_links": {
        "self": {
          "href": "http://example.com/api/author/shahadat"
        }
      },
      "id": "shahadat",
      "name": "Shahadat Hossain Khan",
      "homepage": "http://author-example.com"
    }
  },
  "id": "hal-cookbook",
  "name": "HAL Cookbook"
}
```

General Resource

```
{
  "_links": {
    "self": {
      "href": "http://example.com/api/book/hal-cookbook"
    }
  },
  "id": "hal-cookbook",
  "name": "HAL Cookbook"
}
```

Collections

```
{
  "_links": {
    "self": {
      "href": "http://example.com/api/book/hal-cookbook"
    },
    "next": {
      "href": "http://example.com/api/book/hal-case-study"
    },
    "prev": {
      "href": "http://example.com/api/book/json-and-beyond"
    },
    "first": {
      "href": "http://example.com/api/book/catalog"
    },
    "last": {
      "href": "http://example.com/api/book/upcoming-books"
    }
  },
  "_embedded": {
    "author": {
      "_links": {
        "self": {
          "href": "http://example.com/api/author/shahadat"
        }
      },
      "id": "shahadat",
      "name": "Shahadat Hossain Khan",
      "homepage": "http://author-example.com"
    }
  },
  "id": "hal-cookbook",
  "name": "HAL Cookbook"
}
```

Internet Assigned Numbers Authority - IANA

- Entre as atribuições da IANA, no que diz respeito ao conceito de HATEOAS, destacamos **Atribuição de Protocolo**, que envolve *“o gerenciamento dos parâmetros de protocolo envolve a manutenção de diversos códigos e números utilizados em protocolos de Internet. Isso é feito em conjunto com a IETF (Força-tarefa de Engenharia da Internet)”*.

ICANN (2015)

- Nesse sentido, a IANA define as relações entre o recurso recuperado a partir do serviço e os links definidos por HATEOAS.

Link relations mais comuns

- **Self:** indica que o link aponta para o próprio recurso;
- **Collection:** o link aponta para a coleção do recurso recuperado;
- **Edit:** o link aponta para o destino que é usado para editar o recurso;
- Para saber mais: <https://www.iana.org/assignments/link-relations/link-relations.xhtml>



Prática 01 – Adicionando HATEOAS na sua aplicação

ATENÇÃO!

- Para essa prática, iremos considerar o projeto **MyBooks**, que pode ser baixado no repositório a seguir (***branch main – first commit!***)
- <https://github.com/prof-jlgregorio/mybooks-ads>

Adicionando as dependências

- Abra o arquivo pom.xml e adicione a seguinte dependência:

```
<!-- HATEOAS support -->  
<dependency>  
    <groupId>org.springframework.hateoas</groupId>  
    <artifactId>spring-hateoas</artifactId>  
</dependency>
```

O pacote ***org.springframework.hateoas*** possui as classes necessárias para adicionarmos suporte HATEOAS em nossa aplicação.

Modelo de representação?

- As classes que efetivamente representam os dados da aplicação, ou seja, que determinam os dados retornados nas requisições, seja o Model ou um DTO (Data Transfer Object) devem obrigatoriamente implementar a interface ***org.springframework.hateoas.RepresentationModel***;
- As classes que implementam essa interface possuem métodos que podem ser usados para a geração dos *links* segundo o conceito de HATEOAS;
- Veja como fica a classe model da aplicação:

```
public class CategoryModel extends RepresentationModel implements Serializable {  
    ...  
}
```

Ajustando os métodos do Controller

- Os métodos que precisam de ajustes são os que usam o verbo GET, ou seja, que recuperam dados;
- Por exemplo, o método ***findById*** do controlador retorna um recurso que representa os dados de uma entidade. Logo, por padrão, considerando o HATEOAS, ele precisa retornar um ***link*** para o próprio recurso, o que a IANA chama de ***_self*** (auto link).
- Assim, vamos definir um método chamado ***buildEntityLink*** que recebe um objeto *model* como parâmetro e adiciona o link para ele mesmo. Veja como fica esse método.

```
private void buildEntityLink(CategoryModel category){
    category.add(
        WebMvcLinkBuilder.LinkTo(
            WebMvcLinkBuilder.methodOn(
                CategoryController.class).findById(category.getId())
            ).withSelfRel()
    );
}
```


Entendendo o código

- O método ***add*** está disponível para as classe que implementam a interface ***org.springframework.hateoas.RepresentationModel***.
- Sua função é adicionar **links**, definindo inclusive o link relation, preferencialmente seguindo as definições de IANA;
- Para isso, usamos a classe ***WebMvcLinkBuilder*** que pertence ao pacote ***org.springframework.hateoas.server.mvc***;
- No método ***linkTo*** criamos um **link** apontando para um método do controlador, definido pelo método ***methodOn***;
- Em ***methodOn***, precisamos definir quem é o controlador e o método que desejamos mapear;

Alterando o método findById

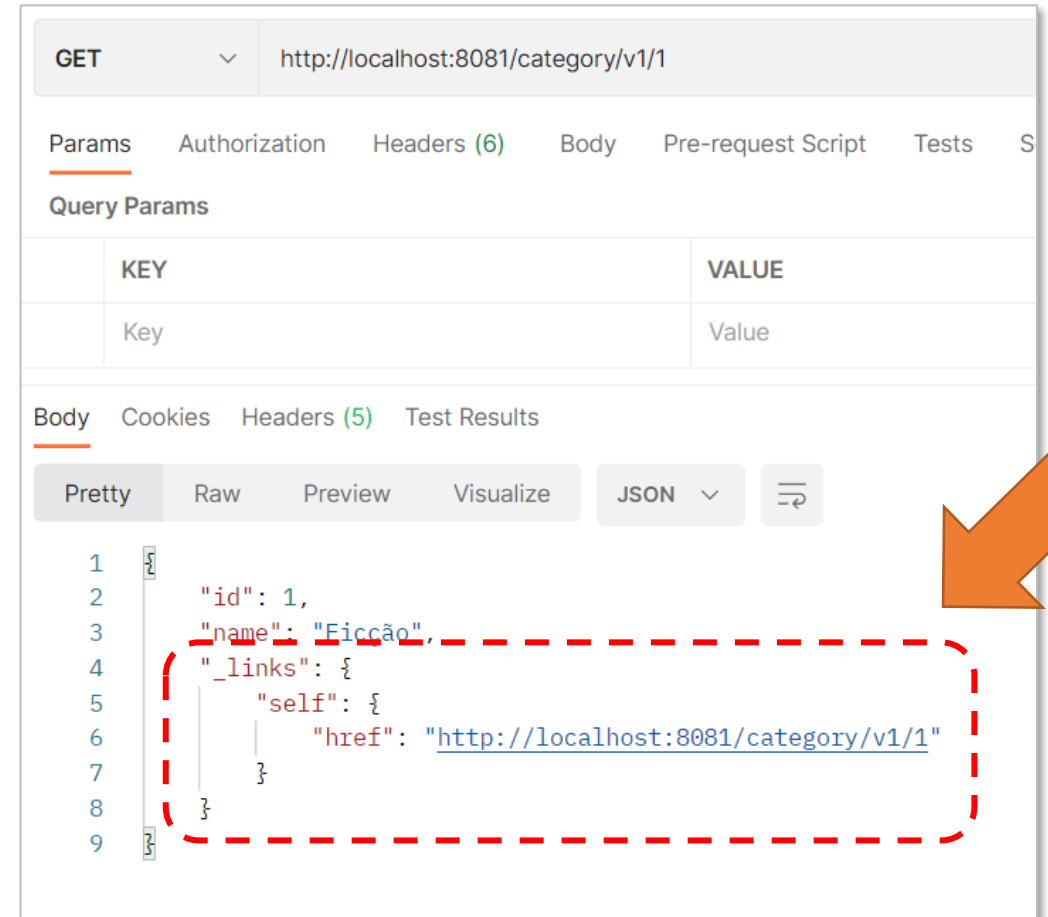
- Note que criamos o link no model referente antes de retorná-lo:

```
@GetMapping(value =("/{id}", produces = {"application/json",  
"application/xml"})  
public CategoryModel findById(@PathVariable Long id){  
    CategoryModel model = service.findById(id);  
    //create the link before return  
    buildEntityLink(model);  
    return model;  
}
```



Testando no Postman

- Ao fazer a busca por categoria, é possível notar que o retorno é um objeto JSON que possui um atributo “**_links**”, que possui um objeto “**self**”, que, por sua vez, possui um atributo “**self**” definindo o *link* para o próprio recurso, definido pelo par chave-valor “**href**” e o *link*.



Link para coleções

- O método ***findAll()*** retorna uma coleção de objetos;
- Portanto, o suporte HATEOAS deve ser adicionado para cada objeto da coleção, e também para a coleção em si;
- O seguinte método ***buildCollectionLink***, recebe um objeto do tipo `CollectionModel` e é usado para a criação de um link para a coleção objetos.
- Note que o *link* para a coleção é criado segundo a especificação IANA.

```
private void buildCollectionLink(CollectionModel<BookModel> books){  
    books.add(WebMvcLinkBuilder.LinkTo(  
        WebMvcLinkBuilder.methodOn(BookController.class).findAll()  
    ).withRel(IanaLinkRelations.COLLECTION));  
}
```

Entendendo o código

- O objeto ***CollectionModel*** informado como parâmetro de entrada se refere a coleção de objetos retornada após a consulta;
- Logo, no método ***findAll***, precisaremos alterar o tipo de retorno para `CollectionModel`;
- A classe `Collection model` faz parte do pacote ***org.springframework.hateoas***
- Objetos `CollectionModel` suportam a criação de *links*, segundo o conceito HATEOAS;

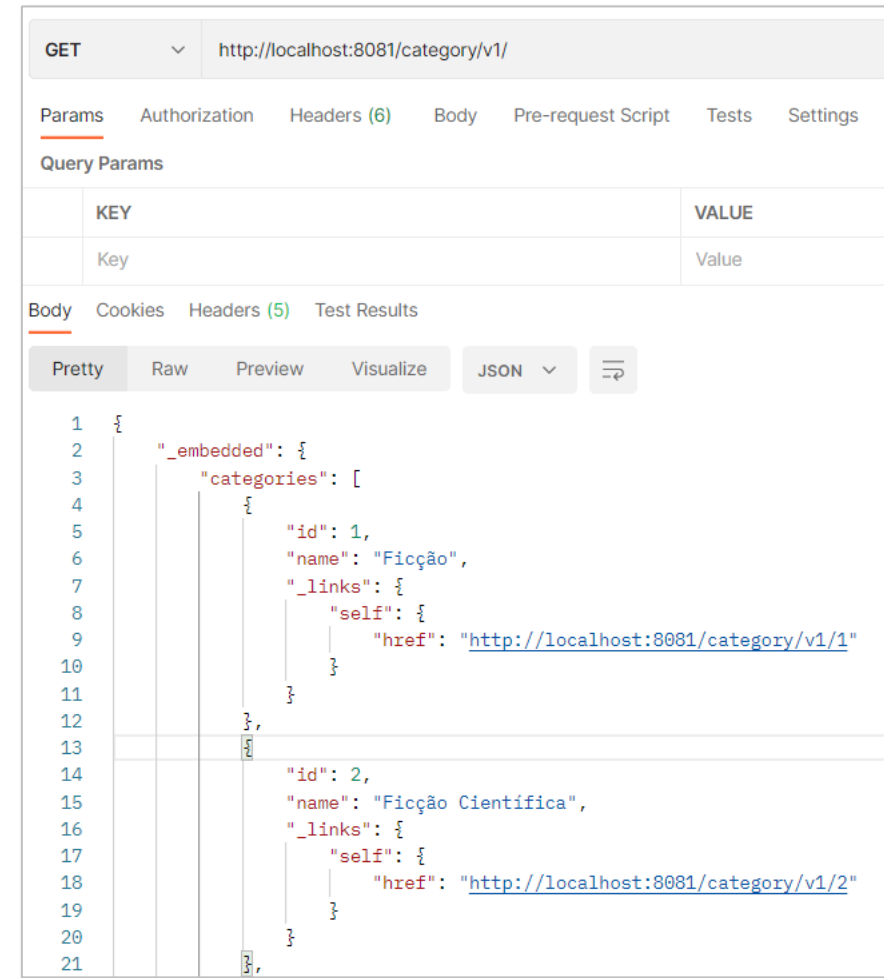
Alterando o método findAll()

- Note as mudanças no método, em destaque:

```
@GetMapping(value = "/", produces = {"application/json", "application/xml"})
public CollectionModel<BookModel> findAll() {
    //..typecast List to CollectionModel
    CollectionModel<BookModel> books = CollectionModel.of(service.findAll());
    //..adding HAL EOAS support for each BookModel
    for(final BookModel book : books){
        buildEntityLink(book);
    }
    //create the link to collection
    buildCollectionLink(books);
    return books;
}
```

Testando no Postman

- Observe que o objeto retornado possui um atributo **_embedded**, que possui o *array* de objetos;
- Cada objeto do *array* possui o *link self*;
- No final, temos o *link* para a *Collection*;




GET <http://localhost:8081/category/v1/>

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

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON 

```
1 {
2   "_embedded": {
3     "categories": [
4       {
5         "id": 1,
6         "name": "Ficção",
7         "_links": {
8           "self": {
9             "href": "http://localhost:8081/category/v1/1"
10          }
11        }
12      },
13      {
14        "id": 2,
15        "name": "Ficção Científica",
16        "_links": {
17          "self": {
18            "href": "http://localhost:8081/category/v1/2"
19          }
20        }
21      }
22    ]
23  }
24 }
```

```
42   "_links": {
43     "collection": {
44       "href": "http://localhost:8081/category/v1/"
45     }
46   }
47 }
```

HATEOAS em relacionamentos

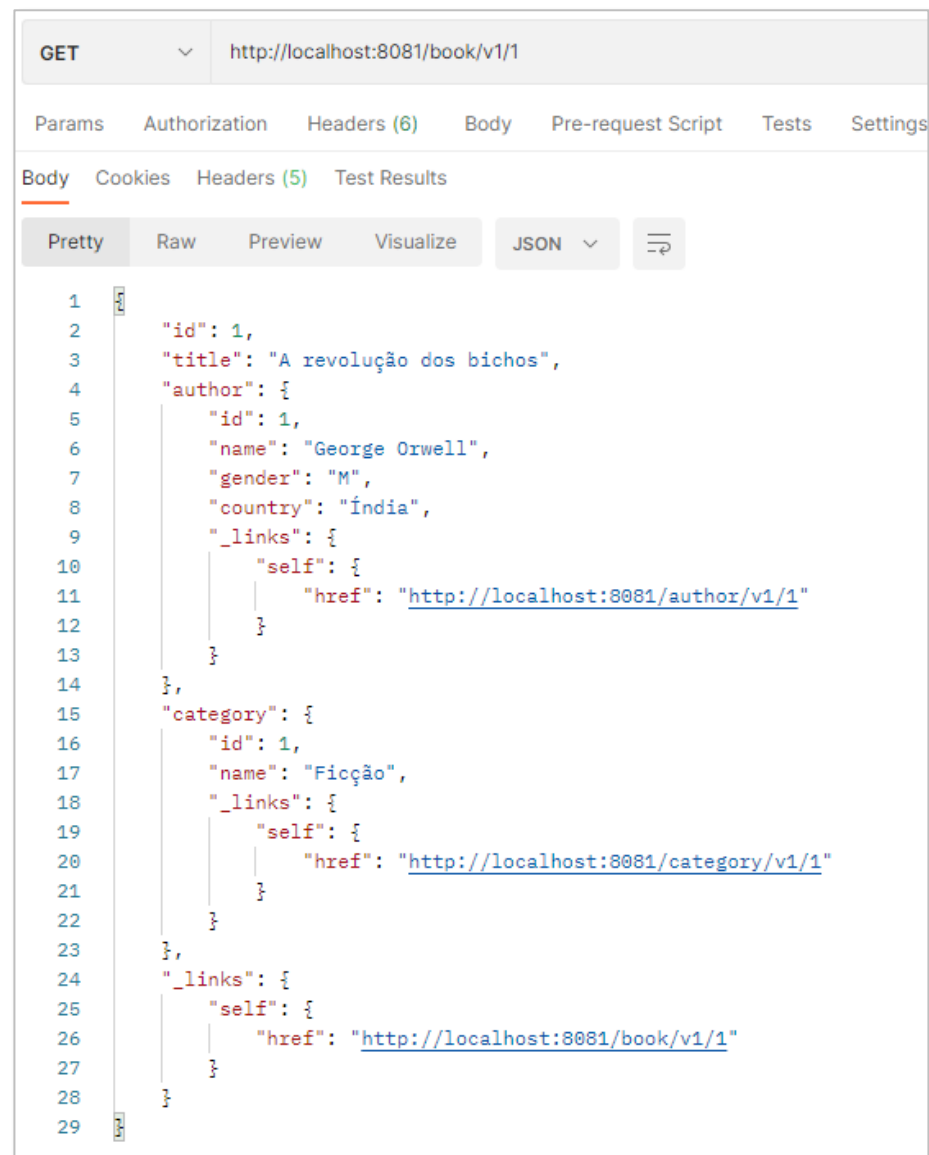
- A entidade **BookModel** possui um relacionamento com as entidades **CategoryModel** e **BookModel**;
- Logo, ao recuperar um **BookModel**, precisamos adicionar os *links* para esses relacionamentos;
- O método **buildEntityLink** recebe um objeto **BookModel** como parâmetro de entrada; Adicionaremos os devidos *links* para esse objeto:
 - *self* → link para o próprio objeto;
 - *category* → link para a categoria;
 - *author* → link para o autor

Adicionando links

```
private void buildEntityLink(BookModel book){  
    //..add a self link  
    book.add(WebMvcLinkBuilder.LinkTo(  
        WebMvcLinkBuilder.methodOn(  
            BookController.class).findById(book.getId())  
        ).withSelfRel());  
    //..add the link of relationships  
    if(!book.getCategory().hasLinks()) {  
        Link categoryLink = WebMvcLinkBuilder.LinkTo(  
            WebMvcLinkBuilder.methodOn(  
                CategoryController.class).findById(book.getCategory().getId()))  
            .withSelfRel();  
        book.getCategory().add(categoryLink);  
    }  
    if(!book.getAuthor().hasLinks()) {  
        Link authorLink = WebMvcLinkBuilder.LinkTo(  
            WebMvcLinkBuilder.methodOn(  
                AuthorController.class).findById(book.getAuthor().getId()))  
            .withSelfRel();  
        book.getAuthor().add(authorLink);  
    }  
}
```

Testando no Postman

- Observe que agora o objeto é retornado com os links para os recursos relacionados;



The screenshot shows the Postman interface for a GET request to `http://localhost:8081/book/v1/1`. The response is displayed in the 'Body' tab, formatted as JSON. The JSON object contains the following data:

```
1  {
2    "id": 1,
3    "title": "A revolução dos bichos",
4    "author": {
5      "id": 1,
6      "name": "George Orwell",
7      "gender": "M",
8      "country": "India",
9      "_links": {
10       "self": {
11         "href": "http://localhost:8081/author/v1/1"
12       }
13     }
14   },
15   "category": {
16     "id": 1,
17     "name": "Ficção",
18     "_links": {
19       "self": {
20         "href": "http://localhost:8081/category/v1/1"
21       }
22     }
23   },
24   "_links": {
25     "self": {
26       "href": "http://localhost:8081/book/v1/1"
27     }
28   }
29 }
```

Link para a Collection

- Assim como fizemos na entidade **CategoryModel**, vamos criar um método para criar o *link* para a coleção e alterar o método *findAll*.
- O método **buildCollectionLink** fica assim:

```
private void buildCollectionLink(CollectionModel<BookModel> books){  
    books.add(WebMvcLinkBuilder.LinkTo(  
        WebMvcLinkBuilder.methodOn(BookController.class).findAll()  
    ).withRel(IanaLinkRelations.COLLECTION));  
}
```


Alterando o método `findAll`

- Agora iremos alterar o método ***findAll*** para que ele retorne um objeto ***CollectionModel***, e não um objeto ***List***:

```
@GetMapping(value = "/", produces = {"application/json", "application/xml"})
public CollectionModel<BookModel> findAll() {
    //..typecast List to CollectionModel
    CollectionModel<BookModel> books = CollectionModel.of(service.findAll());
    //..adding HATEOAS support for each BookModel
    for(final BookModel book : books){
        buildEntityLink(book);
    }
    //create the link to collection
    buildCollectionLink(books);
    return books;
}
```

Métodos com parâmetros opcionais

- Como já visto anteriormente, os ***path params*** são obrigatórios, logo, é mais fácil criar links HATEOAS para eles, visto que sempre são informados;
- Agora, a criação de links para métodos que usam ***query params***, que são opcionais, é um pouco diferente;
- O controlador **BookController** possui um método chamado ***findByTitleOrAuthor***, que possui parâmetros opcionais;
- Esse método recebe ou o parâmetro ***title*** ou o parâmetro ***authorName***;
- Assim para criar os *link* HATEOAS para esse método precisamos considerar a lógica que detecta qual parâmetro foi informado.

Método findByTitleOrAuthor

Note que o tipo de retorno do método foi alterado para ***CollectionModel***.

```
@GetMapping(value = "/find", produces = {"application/json", "application/xml"})
public CollectionModel<BookModel> findByTitleOrAuthor(@RequestParam Optional<String> title,
                                                       @RequestParam Optional<String> authorName) {

    //..creates a list
    List<BookModel> books = new ArrayList<BookModel>();
    //..creates a link
    Link link = null;
    //..if the parameter 'title' is present, perform the search by title
    if (title.isPresent()) {
        books = service.findByTitle(title.get());
        //..the link is defined
        link = WebMvcLinkBuilder.LinkTo(BookController.class)
            .slash("?title="+title.get()).withRel("query");
    }
    //..if the parameter 'authorName' is present, perform the search by authorName
    if (authorName.isPresent()) {
        books = service.findByAuthor(authorName.get());
        //..the link is defined with parameter
        link = WebMvcLinkBuilder.LinkTo(BookController.class)
            .slash("?authorName="+authorName.get()).withRel("query");
    }
    //..iterate the books to create the links
    for (final BookModel bookModel : books) {
        buildEntityLink(bookModel);
    }
    //..create the CollectionModel
    CollectionModel<BookModel> bookCollection = CollectionModel.of(books);
    //..create the link to collection
    buildCollectionLink(bookCollection);
    //..add the link to parametrized method
    bookCollection.add(link);
    //..returns the collection
    return bookCollection;
}
```

Testando no Postman

GET <http://localhost:8081/book/v1/find/?authorName=Ernest>

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

Query Params


KEY	VALUE
<input type="checkbox"/> title	Armada

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ⌵ ≡

```
1  {
2    "_embedded": {
3      "books": [
4        {
5          "id": 9,
6          "title": "Armada",
7          "author": {
8            "id": 4,
9            "name": "Ernest Cline",
10           "gender": "M",
11           "country": "Estados Unidos",
12           "_links": {
13             "self": {
14               "href": "http://localhost:8081/author/v1/4"
15             }
16           }
17         },
18         "category": {
19           "id": 4,
20           "name": "Ficção-Espacial",
21           "_links": {
22             "self": {
23               "href": "http://localhost:8081/category/v1/4"
24             }
25           }
26         },
27         "_links": {
28           "self": {
29             "href": "http://localhost:8081/book/v1/9"
30           }
31         }
32       ]
33     }
34   }
```

```
64   "_links": {
65     "collection": {
66       "href": "http://localhost:8081/book/v1/"
67     },
68     "query": {
69       "href": "http://localhost:8081/book/v1?authorName=Ernest"
70     }
71   }
72 }
```

A decorative graphic on the left side of the slide, consisting of several overlapping, stylized green leaves or petals. The leaves are a light green color and have a smooth, curved shape. They are arranged in a way that they appear to be part of a larger plant or flower, with some leaves pointing upwards and others downwards.

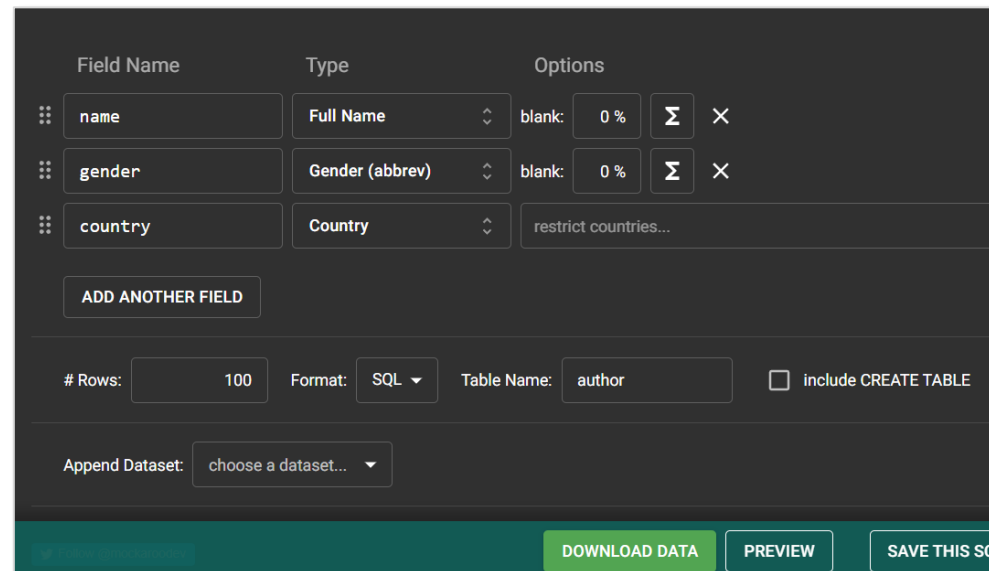
Prática 2 – Paginação de Resultados

Por que paginar resultados?

- Dependendo de como o sistema é pensado, o ciclo ***Request/Response*** pode sobrecarregar o servidor e prejudicar a experiência do usuário nos sistemas clientes, considerando a lentidão para processar grandes quantidades de dados;
- Assim, paginar resultados nas operações GET, que envolvem pesquisas e consultas parametrizadas é imprescindível para sua API de serviços;
- Nas aplicações clientes, a paginação pode ser executada de maneira simples, usando o padrão **Pagination** ou o padrão **Continuous Scrolling**.

Criando dados “mockados”

- O site <https://www.mockaroo.com/> permite a criação de dados fictícios para fins de teste;
- Com ele podemos gerar dados em diversos formatos, inclusive JSON e até gerar instruções SQL para inserção destes dados;
- Ele é muito simples de usar: basta criar os campos, definir os tipos de dados, a quantidade de registros e formato;
- Depois, basta clicar em “**Download Data**” para baixar o dados Mockados;
- No exemplo ao lado, criamos 100 registros para autores fictícios (tabela ***author***);
- Note que não definimos o campo ***id***, pois é Gerado automaticamente;
- Ao ser baixado, note que o arquivo é um conjunto de instruções ***insert into...***
- Execute-o no seu gerenciador de banco de dados;
- **Se preferir, adicione esse arquivo nas *migrations* do seu projeto e execute o commando apropriado para que os dados sejam inseridos**



The screenshot shows the Mockaroo web interface for generating mock data. It features a table configuration section with three fields: 'name' (Full Name), 'gender' (Gender (abbrev)), and 'country' (Country). Each field has a 'blank' percentage set to 0% and a 'restrict' option. Below the table configuration is an 'ADD ANOTHER FIELD' button. The bottom section shows the output settings: '# Rows' set to 100, 'Format' set to SQL, 'Table Name' set to 'author', and an 'Include CREATE TABLE' checkbox. An 'Append Dataset' dropdown is also present. At the bottom right, there are three buttons: 'DOWNLOAD DATA' (highlighted in green), 'PREVIEW', and 'SAVE THIS SC'.

Field Name	Type	Options
name	Full Name	blank: 0 % Σ ×
gender	Gender (abbrev)	blank: 0 % Σ ×
country	Country	restrict countries...

ADD ANOTHER FIELD

Rows: 100 Format: SQL Table Name: author ☐ Include CREATE TABLE

Append Dataset: choose a dataset...

DOWNLOAD DATA PREVIEW SAVE THIS SC

O que preciso mudar?

- Precisaremos mudar os métodos ***find*** na camada controller e também da camada services;
- Na classe ***AuthorController***, procure o método ***findAll***;
- Precisamos alterar o tipo de retorno, pois agora, esse método deverá retornar um objeto ***ResponseEntity*** do tipo ***PageModel*** que é um conjunto de ***AuthorModel***, que é capaz de limitar a quantidade de dados retornados;
- Vamos adicionar dois parâmetros de entrada a esse método:
 - ***page: int*** → determina a página do resultado;
 - ***size: int*** → a qtde de registros que será recuperada em cada página;
 - ***direction: String*** → a direção da ordenação dos dados (“asc” ou “desc”);
 - ***assembler***: objeto do tipo ***PageResourcesAssembler***, responsável por criar os *links* de navegação (*first*, *previous*, *next*, *last*), além de gerar outras informações sobre a paginação;
- O método completo deve ficar assim....

As mudanças estão destacadas com comentários ou pontilhados.

```
@GetMapping(produces = {"application/json", "application/xml", "application/x-yaml"})
public ResponseEntity<PagedModel<AuthorModel>> findAll(@RequestParam(value = "page", defaultValue = "0") int page,
    @RequestParam(value = "size", defaultValue = "10") int size,
    @RequestParam(value = "direction", defaultValue = "asc") String direction,
    PagedResourcesAssembler<AuthorModel> assembler){

    //add a Direction to sort the results
    var sortDirection = "desc".equalsIgnoreCase(direction) ? Sort.Direction.DESC : Sort.Direction.ASC;

    //add a Pageable object to paginate the results
    Pageable pageable = PageRequest.of(page, size, Sort.by(sortDirection, "name"));

    Page<AuthorModel> authors = service.findAll(pageable);
    for (AuthorModel author : authors){
        buildEntityLink(author);
    }

    //returns the ResponseEntity with the links to pagination and HTTP Status OK.
    return new ResponseEntity(assembler.toModel(authors), HttpStatus.OK);
}
```

Alterando a camada Service

- Ao alterar o método da camada Controller, será gerado um erro, pois é necessário que o método ***findAll()*** da camada *service* receba um objeto Pageable.
- Assim, a seguinte modificação será necessária:
 - O retorno do método deve ser um objeto ***Page*** (*org.springframework.data.domain.Page*)
 - O parâmetro de entrada deve ser um objeto ***Pageable*** (*org.springframework.data.domain.Pageable*)
- O código completo deverá ficar assim:

```
public Page<AuthorModel> findAll(Pageable pageable){  
    return repository.findAll(pageable);  
}
```

Alterando o Repository

- Depois que a modificação da camada service for efetivada, o erro acontecerá no Repository;
- Isso quer dizer que o método ***findAll()*** da camada *repository* também precisa de modificações, assim como a camada service.
- Assim, o código completo ficará da seguinte maneira:

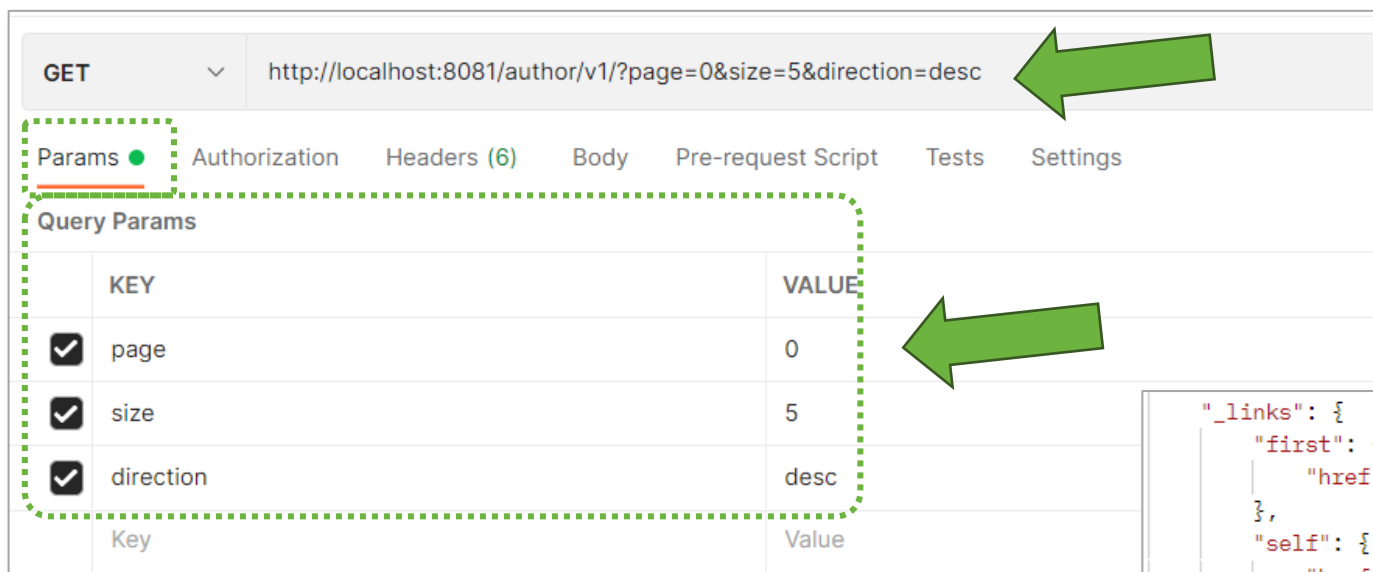
```
//..now, the method receives a Pageable object as input parameter
```

```
public Page<AuthorModel> findAll(Pageable pageable);
```

E o HATEOAS?

- O método ***findAll()*** da camada *controller* não precisa mais método específico para o HATEOAS, pois o método da paginação já gera um link SELF, além dos demais links para paginação de resultados.
- Assim, os links HATEOAS deverão ser gerados manualmente para outras funcionalidades.

Testando no Postman



Links de Navegação



Conclusão

- ***HATEOAS – Hypermedia As The Engine Of Application State*** é um poderoso conceito das aplicações REST;
- O fato de um recurso (resposta) trazer os *links* para diversas funcionalidades associadas a ele mesmo, faz com que as aplicações clientes consigam utilizar os recursos da API de serviços de maneira mais fácil e mais dinâmica, evitando requisições adicionais que envolvam funcionalidades redundantes ou desnecessária em sua aplicação;
- Não dá pra pensar numa API de serviços sem HATEOAS!

Referências

- INTERNET CORPORATION FOR ASSIGNED NAMES AND NUMBERS – ICANN. **As funções da IANA:** Uma introdução às funções da IANA (Autoridade para Atribuição de Números da Internet). 2015. Disponível em: <<https://www.icann.org/pt/system/files/files/iana-functions-18dec15-pt.pdf>>. Acesso em 10 jan. 2021.
- INTERNET ASSIGNED NUMBERS AUTHORITY – IANA. **Link Relations**. 2020. Disponível em: <<https://www.iana.org/assignments/link-relations/link-relations.xhtml>>. Acesso em: 10 jan. 2021.

Sobre mim



JORGE LUÍS GREGÓRIO

- Professor da Faculdade de Tecnologia “Prof. José Camargo” – Fatec Jales, e da Escola Técnica Estadual Dr. José Luiz Viana Coutinho – Etec Jales;
 - Articulista do Jornal de Jales – Coluna “Fatecnologia”;
 - Apresentador do Tech Trends, podcast oficial da Fatec Jales;
 - Bacharel em Sistemas de Informação; Especialista em Desenvolvimento de Software para Web e Mestre em Ciência da Computação.
 - Trabalha com tecnologia desde 1998, tendo atuado como analista de suporte; administrador de redes de computadores; desenvolvedor de software, *webdesigner* e professor.
-
- Site oficial: www.jlgregorio.com.br
 - Perfil do LinkedIn: www.linkedin.com/in/jlgregorio81
 - Currículo Lattes: <http://lattes.cnpq.br/3776799279256689>