



Introdução a Web Services REST com Spring Framework

Parte 4 → Versionando API de serviços

Prof. Me. Jorge Luís Gregório

www.jlgregorio.com.br



Agenda

- Sobre o versionamento da API de serviços;
- Alterando a Model;
- Versionando Services;
- Versionando Controller;
- Testando
- Desafios

A large, stylized green leaf graphic is positioned on the left side of the slide, partially overlapping the title box. It has a light green fill and a darker green outline, with a central vein and several smaller veins branching out.

Sobre o versionamento da API de Serviços

Iniciando o serviço e criando o banco de dados

- É comum a API de serviços evoluir com o tempo;
 - Adicionar ou remover atributos à entidade e ao banco de dados;
 - Adicionar ou remover funcionalidades;
 - Mudanças de requisitos ou regras de negócios;
 - Etc...
- Assim, é necessário criar versões dos ***end-points*** com o objetivo de manter o sistema funcionando, evitando a indisponibilidade do serviço;
- Quando todos os clientes forem informados das mudanças, é recomendado que versões anteriores sejam desativadas;

Como fazer?

- Existem diversas formas, uma delas consiste em **duplicar os recursos** da API de serviços usando o sufixo v1, v2, v3 e assim por diante;
- Assim, se há um model **PersonV1**, podemos criar um **PersonV2**, por exemplo;
- Da mesma forma, se há um **PersonServiceV1**, podemos criar um **PersonServiceV2**;
- Até podemos versionar métodos, mas o mais recomendado, por questões de organização de código, é duplicar os recursos;

A large, stylized green leaf graphic is positioned on the left side of the slide, partially overlapping the green rectangle. It has a light green fill and a darker green outline, with a central vein and several smaller veins branching out.

Alterando a model

Exemplo prático

- Para o nosso exemplo, vamos criar uma nova versão da model Person, adicionando dois atributos: registerDate e gender;
- Abra a classe Person e edicione os campos de acordo com Código ao lado;
- Note que não adicionamos o parâmetro **nullable** nos atributos, visto que a versão anterior da API de serviços desconsidera eles;
- Atualize o **construtor**;
- Adicione os métodos **setters** e **getters**;
- Ao executar a aplicação, como houve alterações na DDL do banco de dados, o Spring irá criar a base novamente, excluindo os registros cadastrados;

```
@Column(name = "register_date")  
private Date registerDate;  
  
@Column(length = 1)  
private String gender;
```

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

Versionando Services

Criando o Service V2

- Copie e cole a classe PersonService e renomeie o novo arquivo para PersonServiceV2;
- Alterações:
 - O método ***save(Person person)*** deverá pegar a data do servidor e atribuir no campo **registerDate**;
 - O método ***update(Person person)*** deverá invocar o método ***setGender(String gender)*** pegando o gênero do objeto passado por parâmetro;
 - Veja o Código ao lado:

```
public Person save(Person person){  
    //set current date  
    person.setRegisterDate(new Date());  
    return repository.save(person);  
}
```

```
public Person update(Person person) throws Exception{  
    Person p = repository.findById(person.getId()).orElseThrow(  
        () → new Exception("Not Found"));  
    p.setFirstName(person.getFirstName());  
    p.setLastName(person.getLastName());  
    p.setProfession(person.getProfession());  
    p.setGender(person.getGender());  
    // don't update registerDate  
    return repository.save(p);  
}
```

Criando o Service V2 – adicionando consulta por nome

- Vamos adicionar uma consulta pelo atributo **firstName**, em que o resultado é uma lista de objetos Person;
- Precisamos usar o conceito de **HQL – Hibernate Query Language**;
- Declare um atributo privado do tipo **EntityManager** (javas.persistence), nomeie-o como **em** e use a anotação **@Autowired** para injeção de dependência (Código ao lado);
- Adicione o método **listByName(String firstname)**
- Veja o código ao lado;

```
@Autowired  
private EntityManager em;
```

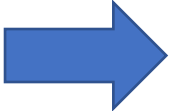
```
public List<Person> listByName(String firstName){  
    String hql = "from Person where first_name like " +  
        " :name order by first_name";  
    Query query = em.createQuery(hql);  
    query.setParameter(name: "name", value: firstName + "%");  
    List<Person> people = query.getResultList();  
    return people;  
}
```

A large, stylized green leaf graphic is positioned on the left side of the slide, partially overlapping the green box. It has a light green fill and a darker green outline, with a central vein and several smaller veins branching out.

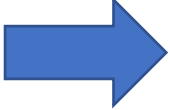
Versionando o Controller

Criando o Controller V2

- A primeira coisa a se fazer é mudar o endpoint do PersonController adicionando o sufixo **/v1** – ver Código ao lado:
- Agora copie e cole a classe PersonController, mudando o nome para PersonControllerV2;
- Mude o **endpoint** adicionando o sufixo /v2 (ao lado)
- Na injeção de dependência, isto é a anotação @Autowired, precisamos atualizar a classe service para PersonServiceV2 (ao lado)



```
@RestController
@RequestMapping("/people/v1")
public class PersonController {
```



```
@RestController
@RequestMapping("/people/v2")
public class PersonControllerV2 {

    @Autowired
    private PersonServiceV2 service;
```

Adicionando o método listByName ao Controller V2

- Vamos adicionar um novo endpoint com o sufixo */query* em que passamos uma *Path Variable* informando o nome que queremos pesquisar.
- Veja o código a seguir:

```
@GetMapping("/query/{firstName}")  
//the '/query' sufix is used to avoid conflict with others get mappings  
public List<Person> listByName(@PathVariable("firstName") String firstName){  
    return service.listByName(firstName);  
}
```

A large, stylized green leaf graphic is positioned on the left side of the slide, partially overlapping the title box. It has a light green fill and a darker green outline, with a central vein and several smaller veins branching out.

Testando no Postman

Testando o verbo *post* das duas versões

POST ▼ http://localhost:8080/people/v1 Send ▼

Params Auth Headers (8) Body ● Pre-req. Tests Settings Cookies Beautify

raw ▼ JSON ▼

```
1 {
2   ... "firstName": "Jorge",
3   ... "lastName": "Gregório",
4   ... "profession": "Professor"
5 }
```

Body Cookies Headers (5) Test Results 200 OK 124 ms 274 B Save Response ▼

Pretty Raw Preview Visualize JSON ▼ ≡

```
1 {
2   "id": 1,
3   "firstName": "Jorge",
4   "lastName": "Gregório",
5   "profession": "Professor",
6   "registerDate": null,
7   "gender": null
8 }
```

POST ▼ http://localhost:8080/people/v2 Send ▼

Params Auth Headers (8) Body ● Pre-req. Tests Settings Cookies Beautify

raw ▼ JSON ▼

```
1 {
2   ... "firstName": "Patricia",
3   ... "lastName": "Gregório",
4   ... "profession": "Professora",
5   ... "gender": "F"
6 }
```

Body Cookies Headers (5) Test Results 200 OK 60 ms 304 B Save Response ▼

Pretty Raw Preview Visualize JSON ▼ ≡

```
1 {
2   "id": 2,
3   "firstName": "Patricia",
4   "lastName": "Gregório",
5   "profession": "Professora",
6   "registerDate": "2021-11-19T12:52:34.830+00:00",
7   "gender": "F"
8 }
```

Testando a consulta por nomes

GET `http://localhost:8080/people/v2/query/Jo` [Send](#)

Params Auth Headers (8) **Body** Pre-req. Tests Settings Cookies Beautify

raw JSON

```
1 {
2   ... "firstName": "José",
3   ... "lastName": "Silva",
4   ... "profession": "Mecânico",
5   ... "gender": "M"
6 }
```

Body Cookies Headers (5) Test Results 200 OK 15 ms 409 B [Save Response](#)

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "id": 1,
4     "firstName": "Jorge",
5     "lastName": "Gregório",
6     "profession": "Professor",
7     "registerDate": null,
8     "gender": null
9   },
10  {
11    "id": 3,
12    "firstName": "José",
13    "lastName": "Silva",
14    "profession": "Mecânico",
15    "registerDate": "2021-11-19T14:25:50.000+00:00",
16    "gender": "M"
17  }
18 }
```

GET `http://localhost:8080/people/v2/query/Joao` [Send](#)

Params Auth Headers (8) **Body** Pre-req. Tests Settings Cookies Beautify

raw JSON

```
1 {
2   ... "firstName": "José",
3   ... "lastName": "Silva",
4   ... "profession": "Mecânico",
5   ... "gender": "M"
6 }
```

Body Cookies Headers (5) Test Results 200 OK 7 ms 166 B [Save Response](#)

Pretty Raw Preview Visualize JSON

```
1 {
2 }
```


Conclusão

- O versionamento de API de Serviço é algo que deve ser considerado desde o início do projeto;
- O principal motivo para possibilitar o uso de diferentes versões da API é manter o serviço sempre disponível para os clientes, mesmo quando o projeto está passando por atualizações;
- Note que no exemplo deste material, não criamos novas versões das classes **model** e **repository**, porém, dependendo da complexidade das mudanças, essas são alterações que poderiam ser realizadas;
- Considere criar pacotes para cada uma das versões criadas;
- Baixe o código!
 - O projeto trabalhado nesse material está disponível no GitHub no link a seguir: <https://github.com/prof-jlgregorio/spring-crud-mysql>
 - Note que há dois branches: main e crud-v2.
 - No *branch* **main** está o projeto antes das alterações;
 - No *branch* **crud-v2** está todas as alterações realizadas neste material.

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>

