Residência em Software

# Módulo Programação JAVA (Avançado)

MÊS 01

INSTITUIÇÃO EXECUTORA

cepedi UESC

COORDENADORA

MCTI FUTURO Softex

APOIO

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO

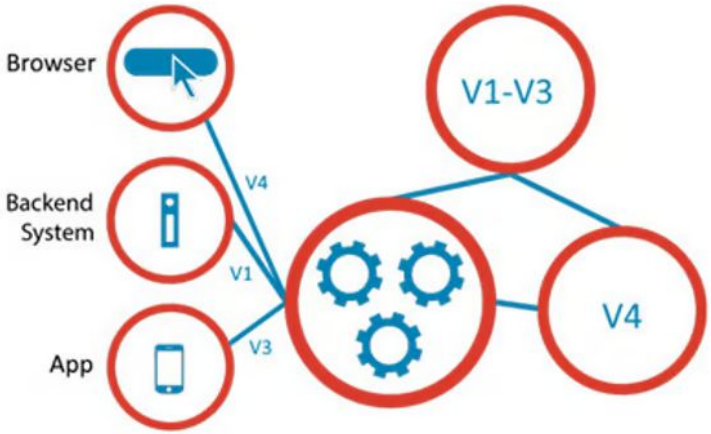
GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

Residência em Software

# Versionamento de APIs com **Spring Boot**

## Versionamento de APIs

O controle de versão de API é um aspecto crucial do desenvolvimento de software, que permite **introduzir novos recursos** sem interromper a funcionalidade existente, especialmente quando precisamos **manter e estender** uma base de código existente.

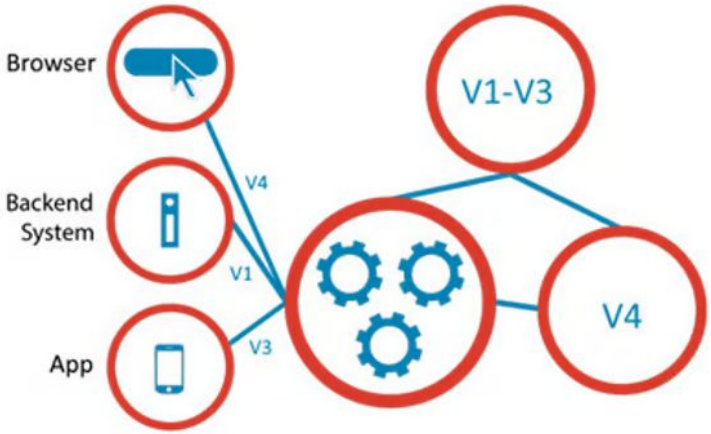


Residência em Software

# Versionamento de APIs com **Spring Boot**

## Versionamento de APIs

A incorporação do controle de versão da API é uma prática recomendada que contribui para a capacidade de manutenção e confiabilidade do software ao longo do tempo, garantindo que você possa continuar a atender às necessidades de seus usuários e, ao mesmo tempo, minimizar as interrupções nos sistemas existentes.



# Versionamento de APIs com **Spring Boot**

## Versionamento de APIs

Muitos dos serviços mais populares, como X(“Twitter”), Facebook, Netflix ou PayPal estão versionando suas APIs REST.
As vantagens e desvantagens dessa abordagem são óbvias. Por um lado, você **não precisa se preocupar em fazer alterações em sua API**, mesmo que muitos clientes e aplicativos externos a consumam.
Por outro lado, você **precisa manter diferentes versões de implementação de API** em seu código, o que às vezes pode ser problemático.

# Versionamento de APIs com **Spring Boot**

## Benefícios do Versionamento de APIs

O versionamento de API é uma prática fundamental no desenvolvimento de software, especialmente em um mundo onde as aplicações estão em constante evolução. Entre os seus benefícios, incluem:

**Compatibilidade com versões anteriores:** ao controlar o controle de versão de suas APIs, você garante que os clientes e sistemas existentes continuem funcionando conforme o esperado. Isso significa que você pode introduzir novos recursos ou fazer alterações sem interromper os usuários da versão anterior.

# Versionamento de APIs com **Spring Boot**

## Benefícios do Versionamento de APIs

**Transição mais suave:** o controle de versão permite uma transição gradual e controlada entre as versões. Ele dá aos desenvolvedores a flexibilidade de atualizar seus sistemas em seu próprio ritmo, reduzindo o risco de mudanças abruptas e potencialmente perturbadoras.

# Versionamento de APIs com **Spring Boot**

## Benefícios do Versionamento de APIs

**Documentação mais limpa:** o controle de versão da API fornece uma maneira clara de documentar e comunicar alterações. Os clientes podem consultar versões específicas da API para entender o que é suportado e o que pode ter mudado.

# Versionamento de APIs com **Spring Boot**

## Benefícios do Versionamento de APIs

**Testes aprimorados:** versões separadas da API tornam o teste e a garantia de qualidade mais gerenciáveis. Ele permite testes focados nas mudanças específicas de uma determinada versão, reduzindo o risco de regressões.

# Versionamento de APIs com **Spring Boot**

## Benefícios do Versionamento de APIs

**Preparação para o futuro:** o controle de versão permite que você planeje o futuro. Você pode projetar sua API com o conhecimento de que ela evoluirá, facilitando a adaptação às mudanças nos requisitos e ao feedback dos clientes.

# Versionamento de APIs com **Spring Boot**

## Desafios no Versionamento de APIs

A implementação do controle de versão também traz consigo **desafios potenciais**, incluindo problemas relacionados a alterações de esquema de banco de dados e consistência de dados. Podemos citar como exemplo:

**Alterações no esquema do banco de dados:** se as versões da API envolverem alterações no esquema de banco de dados subjacente, como adicionar ou remover colunas, isso poderá causar problemas de controle de versão do banco de dados. Versões de API diferentes podem precisar de estruturas de banco de dados diferentes, o que pode ser complexo de gerenciar.

# Versionamento de APIs com **Spring Boot**

## Desafios no Versionamento de APIs

**Migração de dados:** quando você modifica o esquema de banco de dados, geralmente precisa executar migrações de dados para garantir que os dados existentes estejam conforme o novo esquema. Esse processo pode ser propenso a erros, demorado e pode exigir um planejamento cuidadoso.

# Versionamento de APIs com **Spring Boot**

## Desafios no Versionamento de APIs

**Consistência dos dados:** em um ambiente de API de várias versões, garantir a consistência dos dados entre as versões pode ser um desafio. Por exemplo, se a versão 1 e a versão 2 da API manipularem dados de forma diferente, você precisará gerenciar as transições de dados de forma eficaz para evitar inconsistências.

# Versionamento de APIs com **Spring Boot**

## Desafios no Versionamento de APIs

**Maior complexidade:** o gerenciamento de várias versões de API, cada uma com requisitos de banco de dados potencialmente exclusivos, pode adicionar complexidade ao seu aplicativo. Essa complexidade pode tornar a manutenção e a solução de problemas mais desafiadoras.

# Versionamento de APIs com **Spring Boot**

## Desafios no Versionamento de APIs

**Teste de compatibilidade:** testar todas as versões da API e suas interações com o banco de dados pode ser complicado. É imprescindível manter um conjunto abrangente de testes para garantir a compatibilidade entre as versões, requerendo esforço extra.

# Versionamento de APIs com **Spring Boot**

## Desafios no Versionamento de APIs

**Performance Overheads:** o suporte a várias versões de API pode introduzir alguma sobrecarga de desempenho, especialmente se houver extensas transformações de dados ou adaptações a serem feitas entre as versões.

Residência em Software

# Versionamento de APIs com **Spring Boot**

## Estratégias de Versionamento

Ao implementar o controle de versão de APIs em um aplicativo Spring Boot, você pode explorar diferentes estratégias de implementação sem interromper a compatibilidade com clientes existentes. Cada método tem seus prós e contras, então escolha o que melhor se adapta às necessidades do seu projeto.
Examinaremos quatro técnicas comuns de controle de versão: controle de versão de URI, controle de versão de parâmetro de solicitação, controle de versão de cabeçalho personalizado e controle de versão de negociação de conteúdo (aceitar cabeçalho).

Residência em Software

# Versionamento de APIs com **Spring Boot**

## Estratégias de Versionamento — Controle de versão de URI

O controle de versão por caminho de URI envolve a adição do número da versão à URL da API. Este é um método simples e facilmente compreensível. No entanto, pode não ser o mais elegante, por poder levar a URLs longos.

```
@RestController
@RequestMapping("/api/v1/users")
public class UserControllerV1 {

    @GetMapping
    public ResponseEntity<List<User>> getUsers() {
        // Implementation here
    }

    // Other endpoints
}
```

Residência em Software

# Versionamento de APIs com **Spring Boot**

## Estratégias de Versionamento — Controle de versão de parâmetro de solicitação

Com essa abordagem, o número da versão é passado como um parâmetro de solicitação. É menos intrusivo do que o controle de versão de URI, mas pode não ser tão intuitivo para os consumidores.

```
@RestController
public class UserController {

    @GetMapping(value = "/api/users", params = "version=1")
    public ResponseEntity<List<User>> getUsersV1() {
        // Implementation here
    }

    @GetMapping(value = "/api/users", params = "version=2")
    public ResponseEntity<List<User>> getUsersV2() {
        // Implementation here
    }
}
```

Residência em Software

# Versionamento de APIs com **Spring Boot**

## Estratégias de Versionamento — Controle de versão por cabeçalho personalizado

O controle de versão por cabeçalho personalizado envolve a adição de um cabeçalho personalizado à solicitação HTTP, contendo o número da versão. Esse método mantém o URI limpo, mas os consumidores devem se lembrar de incluir o cabeçalho personalizado.

```
@RestController
public class UserController {

    @GetMapping(value = "/api/users", headers = "X-API-Version=1")
    public ResponseEntity<List<User>> getUsersV1() {
        // Implementation here
    }

    @GetMapping(value = "/api/users", headers = "X-API-Version=2")
    public ResponseEntity<List<User>> getUsersV2() {
        // Implementation here
    }
}
```

Residência em Software

# Versionamento de APIs com **Spring Boot**

## Estratégias de Versionamento — Controle de versão por negociação de conteúdo

O controle de versão por negociação de conteúdo aproveita o cabeçalho HTTP Accept para especificar a versão de API desejada.

```
@RestController
public class UserController {

    @GetMapping(value = "/api/users", produces = "application/vnd.example.api.v1+json")
    public ResponseEntity<List<User>> getUsersV1() {
        // Implementation here
    }

    @GetMapping(value = "/api/users", produces = "application/vnd.example.api.v2+json")
    public ResponseEntity<List<User>> getUsersV2() {
        // Implementation here
    }
}
```

# Versionamento de APIs com **Spring Boot**

## Adicionando o Controle de Versão na API

**Etapa 1**: pom.XML

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.8.1</version>
    <configuration>
        <source>21</source> <!-- sua versão do Java -->
        <target>21</target> <!-- sua versão do Java -->
        <compilerArgs>
            <arg>-parameters</arg>
        </compilerArgs>
    </configuration>
</plugin>
```

Residência em Software

# Versionamento de APIs com **Spring Boot**

## Adicionando o Controle de Versão na API

**Etapa 2**: adicionar anotações no ServiceV1 (original)

- https://gist.github.com/rog3r/57763ba9407e2a1f1303114a11ca78d3

**Etapa 3**: criar um novo ServiceV2 (**estendendo** o V1) e criar novas funcionalidades

- https://gist.github.com/rog3r/f601e385eec937ab5642366bf38e4ae4

**Etapa 4**: criar os controladores versionados EmployeeControllerV1 e EmployeeControllerV2

- https://gist.github.com/rog3r/290cf621cfc00a38d128ba4758407f64
- https://gist.github.com/rog3r/d76e323b41b9b7a355bac2cc2fe34dfc

Residência em Software

## Contato

rogerio.jesus@cepedi.org.br

https://moodle.residenciatic18.cepedi.org.br/