

# Aprimorando os Endpoints

Parâmetros, valores de retorno e outros detalhes

# Dificuldade no desenvolvimento de um serviço

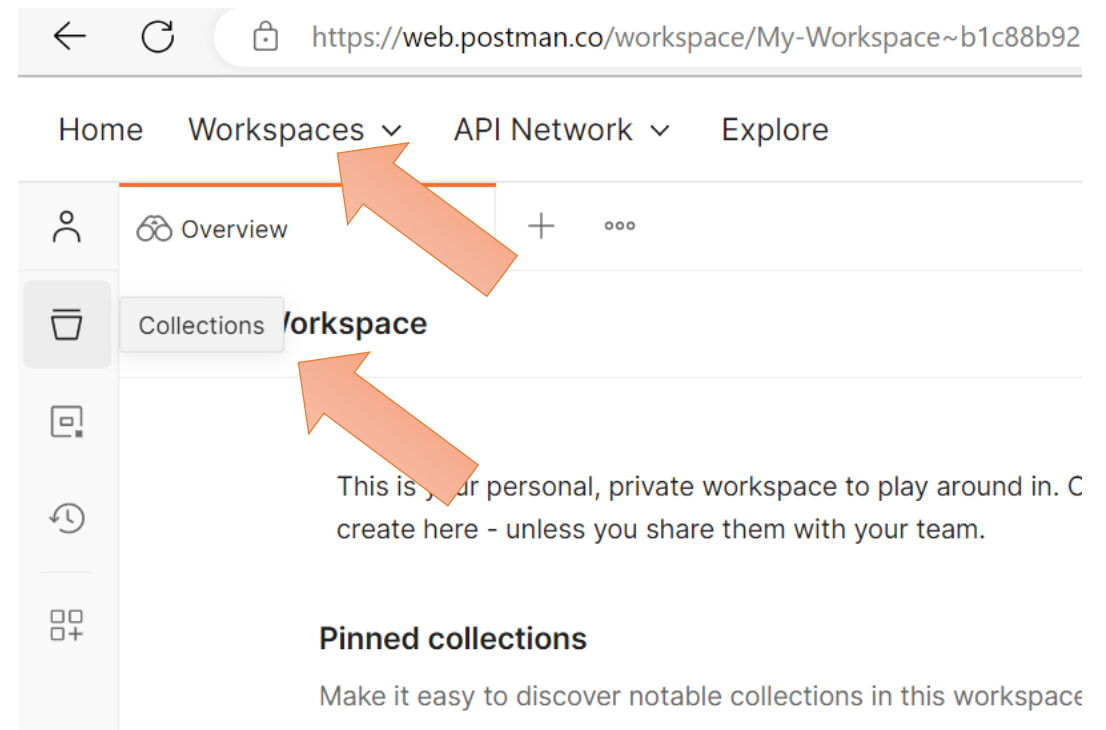
- Na etapa anterior iniciamos o desenvolvimento de um “backend” de uma biblioteca, ou seja, um serviço capaz de responder uma série de requisições HTTP relativas ao acervo de uma biblioteca.
- Um serviço, entretanto, não possui interface com o usuário. Desta forma para testar as funcionalidades oferecidas disparamos requisições HTTP diretamente pelo navegador.
- Ainda que este método funcione para requisições simples, mostra-se pouco prático, na medida que temos de reescrever cada uma das solicitações toda a vez que queremos repetir uma operação. Além disso não existem maiores recursos para acompanhar ou depurar as requisições.

# O software “Postman”

- O Postman é um software que auxilia no desenvolvimento de serviços (APIs) que atendem requisições HTTP.
- Entre os principais recursos destacamos o que permite que sejam definidas coleções de requisições HTTP que podem ser reusadas a qualquer momento.
- O Postman pode ser usado tanto em versão instalada como diretamente no navegador
- Você encontra a ferramenta em [Postman API Platform](#).

# Entendendo o software “postman”

- Acesse a página WEB do Postman
- Crie um usuário e senha (é gratuito)
- Crie um “workspace” para trabalhar
- Crie uma “coleção”
  - Em uma coleção você pode guardar um conjunto de requisições HTTP
  - Crie uma coleção para cada API que for desenvolver
  - Exemplo: crie uma coleção chamada “Biblioteca” para trabalhar com estes primeiros exemplos.

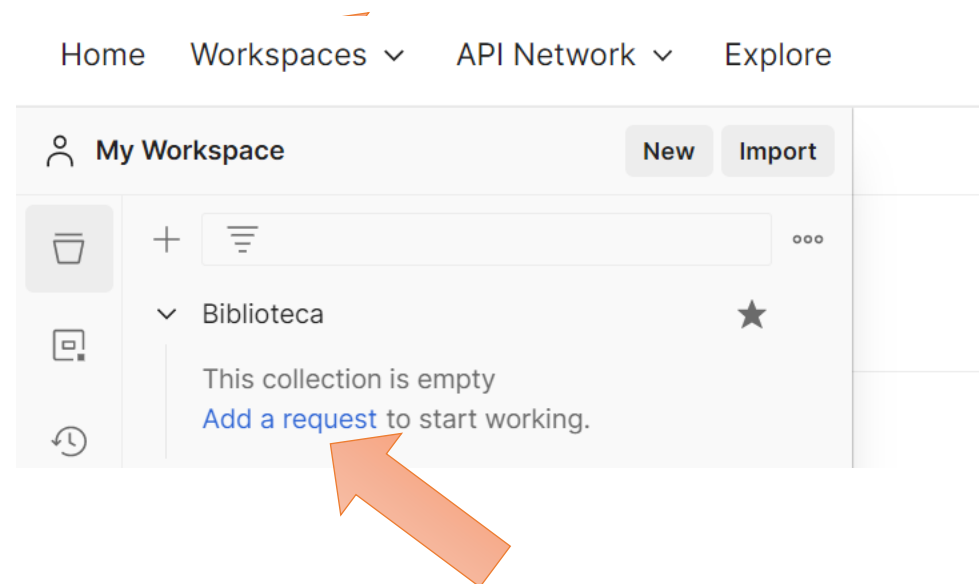


# Antes de continuar ...

- Antes de continuar coloque o serviço de biblioteca para executar (coloque o serviço “no-ar”)
- Não esqueça que nosso objetivo com o Postman é testar o serviço enviando requisições para o mesmo, então é necessário que o servidor WEB esteja executando
- Para disparar o serviço a partir da linha de comando:
  - Mova par a pasta onde estão projeto do serviço
  - Use o comando “mvn spring-boot:run”

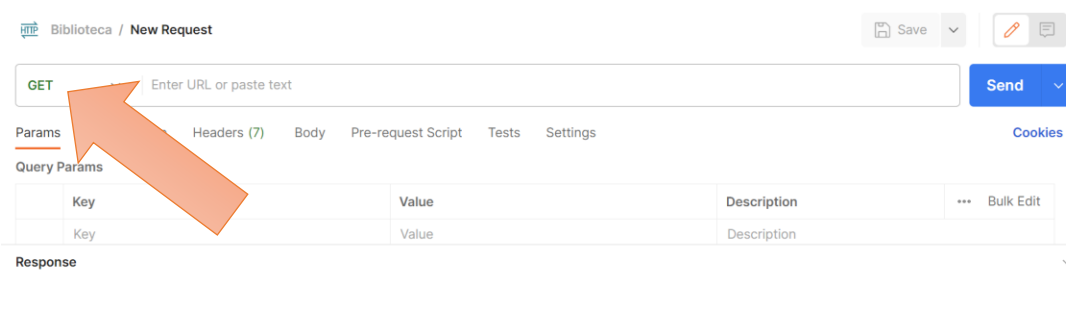
# Criando uma requisição HTTP

- Uma vez criada a biblioteca, adicione uma requisição HTTP



# Criando uma requisição HTTP

- Escolha o tipo de requisição



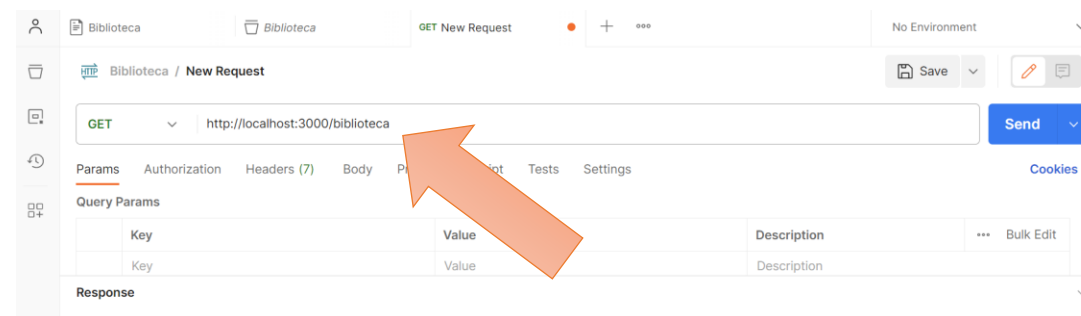
The screenshot shows the 'New Request' interface of an HTTP client. At the top, there's a breadcrumb 'Biblioteca / New Request' and a 'Save' button. Below this is a form with a dropdown menu set to 'GET' and a text input field labeled 'Enter URL or paste text'. A blue 'Send' button is to the right. Below the form are tabs for 'Params', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Params' tab is active, showing a 'Query Params' section with a table. An orange arrow points to the 'GET' dropdown menu.

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Response

# Criando uma requisição HTTP

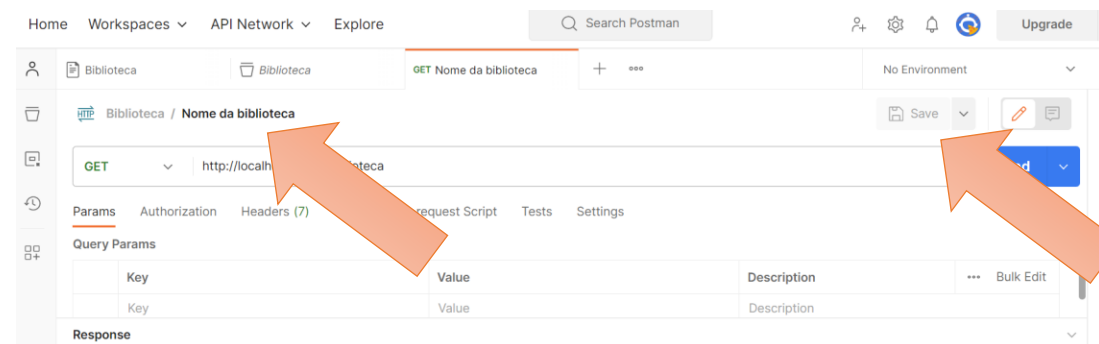
- Escolha o tipo de requisição
- Escreva a requisição propriamente dita





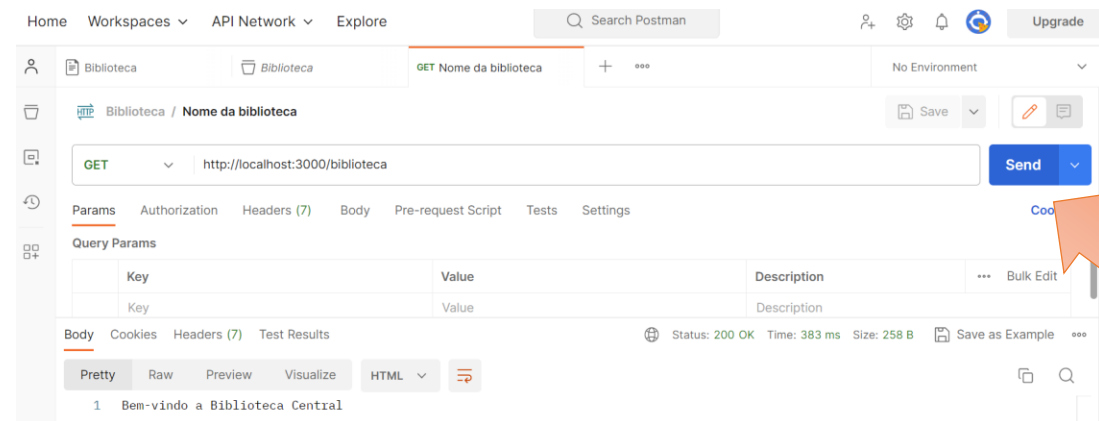
# Criando uma requisição HTTP

- Escolha o tipo de requisição
- Escreva a requisição propriamente dita
- De um nome para a requisição e salve



# Criando uma requisição HTTP

- Escolha o tipo de requisição
- Escreva a requisição propriamente dita
- De um nome para a requisição e salve
- Envie a requisição e observe o resultado



# Uma alternativa ao uso do Postman

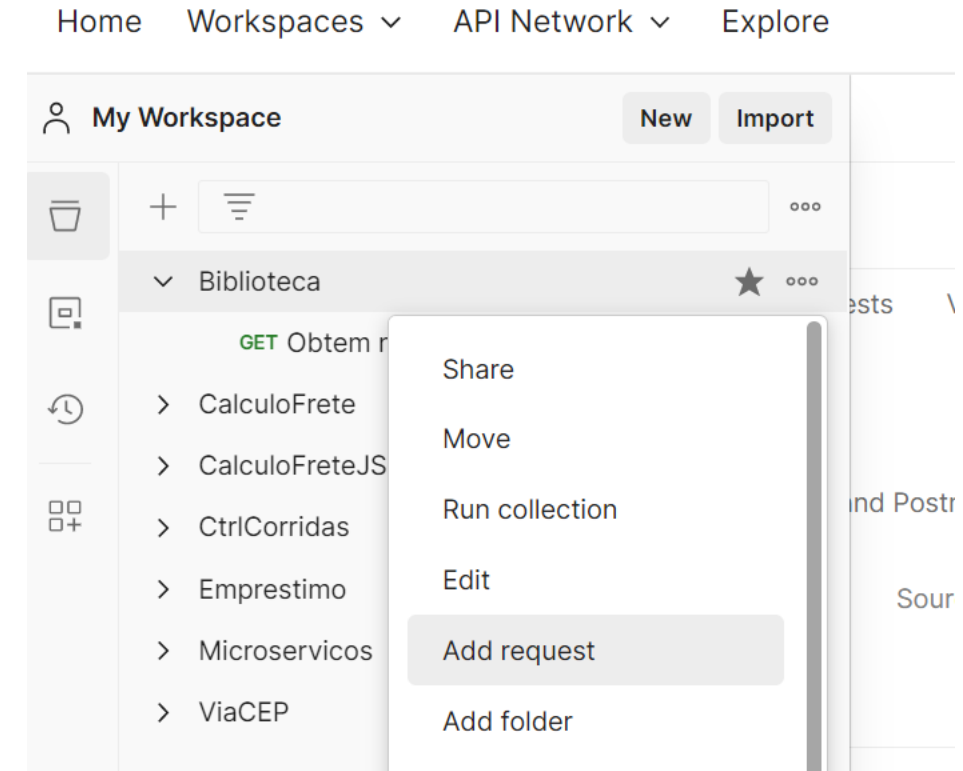
- Alternativamente ao uso do Postman é possível criar arquivos com requisições HTTP dentro do próprio projeto do VSCode
- Para tanto instale a extensão chamada “[Rest Client](#)”
- Crie um arquivo com extensão .http e anote suas requisições neste arquivo (consulte a documentação da extensão)
- OBS: não funciona adequadamente quando executado no VSCode WEB



## Dinâmica

Crie requisições para testar os “endpoints” desenvolvidos na aula anterior:

- <http://localhost:8080/biblioteca/livros>
  - <http://localhost:8080/biblioteca/titulos>
  - <http://localhost:8080/biblioteca/autores>
- 
- Observe, para cada requisição, o tipo e a forma dos resultados obtidos.
  - Teste também uma requisição inválida para verificar os tipos de mensagens de erro



# Valores de retorno de uma requisição HTTP

- Quando se usa o Spring Boot (como visto nos exemplos até aqui) o comportamento dos valores de retorno de uma requisição HTTP seguem as seguintes regras:
  - Se o valor retornado pela função que responde a requisição é um objeto Java ou um arranjo, então eles serão automaticamente serializados para um JSON.
  - Se o valor for um tipo “primitivo” (string, número ou booleano) , então eles serão retornados sem serem serializados em JSON
- E relação ao código de status de resposta, este será sempre 200.

# Passagem de parâmetros

- Até agora trabalhamos com requisições “Get” estáticas, isto é, sem parâmetros. Para que se possa desenvolver aplicações com aplicabilidade prática, porém, é necessário que nossas requisições possam ser mais dinâmicas, permitindo o envio de parâmetros variáveis. O protocolo HTTP permite 3 formas de parametrização:
  - “Query strings” (strings de consulta)
  - “Route parameters” (parâmetros na rota)
  - “Body data” (dados no corpo da mensagem)
- Na sequência iremos analisar exemplos destes 3 casos

# Tratando “Query strings”

- No protocolo HTTP uma “query string” segue o formato:  
`[path["?"<var>=<valor>["&"<var>=<valor>]]`
- Exemplo:  
<http://localhost:8080/biblioteca/livrosautor?autor=Miguel%20de%20Cervantes>
- Para tratar este tipo de requisição usamos a decoração `@RequestParam` como no exemplo abaixo:

```
@GetMapping("/livrosautor")  
public List<Livro> getLivrosDoAutor(@RequestParam String autor) {  
    return livros.stream().filter(l -> l.getAutor().equals(autor)).toList();  
}
```

# Tratando “Path parameters”

- O protocolo HTTP admite parâmetros na própria rota
- Exemplo: <http://localhost:8080/biblioteca/livrostitulo/Dom%20Quixote>
- Para tratar este tipo de requisição usamos a decoração `@PathVariable` como no exemplo abaixo:

```
@GetMapping("/livrostitulo/{titulo}")  
public List<Livro> getLivrosPorTitulo(@PathVariable String titulo) {  
    return livros.stream().filter(l -> l.getTitulo().equals(titulo)).toList();  
}
```

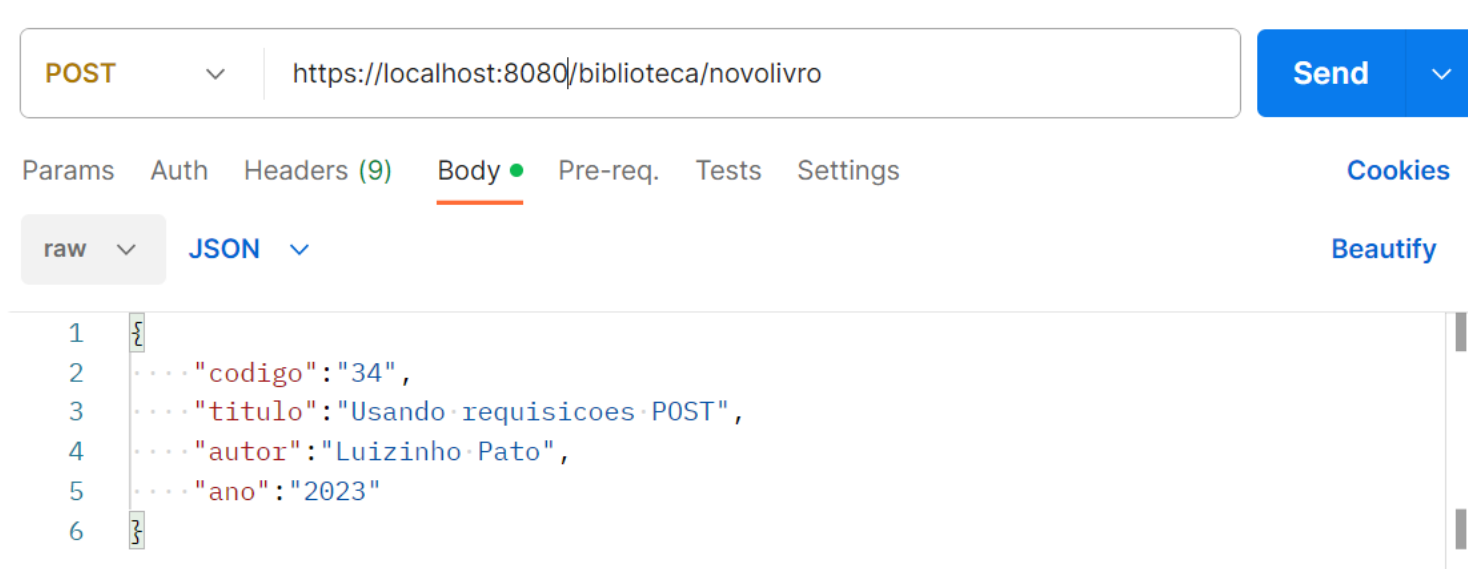


# Dados no corpo da mensagem

- O protocolo HTTP admite que se enviem dados no “corpo” (body) da mensagem
- Normalmente este recurso é usado em requisições do tipo “POST”.
- O importante neste caso é conhecer os atributos do(s) objeto(s) que serão enviados
- Caso contrário será difícil tratar os dados recebidos

# Enviando dados no corpo da mensagem

- Se estiver usando o "postman" para enviar uma requisição POST com dados no corpo da mensagem, especifique o JSON com os dados conforme a figura



# Recebendo dados no corpo da mensagem

- Para receber os dados no corpo da mensagem, use a decoração "@RequestBody" conforme o código abaixo.
- O fundamental neste caso é que os atributos do objeto que será enviado correspondam aos da classe do objeto que recebe os dados.

```
@PostMapping("/novolivro")  
public boolean cadastraLivroNovo(@RequestBody final Livro livro) {  
    livros.add(livro);  
    return true;  
}
```

# Parâmetros de retorno

- Nos exemplos vistos até agora as funções que tratam os endpoints simplesmente retornam os valores desejados e o framework se encarrega da serialização e montagem da resposta HTTP.
- Em algumas situações, porém, pode ser necessário ter mais controle sobre a montagem da resposta de maneira a poder editar os códigos de retorno, cabeçalhos, etc.
- Para tanto pode-se usar a classe *ResponseEntity*.

# Usando a classe *ResponseEntity*

```
@GetMapping("/livrostitulo/{titulo}")
public ResponseEntity<List<Livro>> getLivrosPorTitulo(@PathVariable String
titulo) {
    var resultado = livros.stream().filter(l -> l.getTitulo().equals(titulo)).toList();
    if (resultado.isEmpty()) {
        return ResponseEntity.notFound().build();
    }
    return ResponseEntity.status(HttpStatus.OK).body(resultado);
}
```

- Usando **ResponseEntity** podemos editar todos os atributos da mensagem de resposta via padrão *builder* ou utilizar métodos *factory* que já configuram a mensagem
- Observe, também, como é feita a declaração do tipo de retorno (ResponseEntity é uma classe genérica).
- Consulte a documentação para verificar as outras possibilidades de configuração da mensagem.



## Dinâmica

Acrescente os seguintes caminhos no nosso backend:

- a) `LivrosPorAno?ano=<valor>` → devolve a lista dos livros publicados no ano indicado usando uma query string
- b) `Desatualizados/{ano}` → devolve a lista dos livros cujo ano de publicação é inferior ao ano informado
- c) Crie uma rota que permita consultar todos os livros de um determinado autor publicados em um determinado ano
- d) Crie uma rota “POST” que permita atualizar/corrigir os dados de um livro do acervo da biblioteca (atualiza todos os campos menos o código).

