



Recebendo Requests - Spring Web

Os **requests** que recebemos dependem do contrato que os endpoints possuem. As vezes podemos retornar dados sem esperar nada de quem está fazendo a requisição, mas as vezes **precisamos que algumas informações sejam passadas**.



PathVariable - Spring Web

As informações passadas podem ser através de **PathVariables**, quando usamos essa anotação estamos dizendo que é obrigatório que o **client passe a informação pelo caminho**:

GET `http://localhost:8080/api/goals/1`

```
@GetMapping("/{goalId}")
public ResponseEntity<DetailedGoalResponse> findGoal(@PathVariable Long goalId) {
    Goal goal = goalService.findById(goalId);
    return ResponseEntity.ok(new DetailedGoalResponse(goal));
}
```

```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

Request Line
Request Headers
Request Message Header
A blank line separates header & body
Request Message Body



RequestParam - Spring Web

Outra forma de receber parâmetros é usando o **RequestParam**, a diferença é que esse tipo de parâmetro é **opcional**. Eles são passados após a **"?"** no caminho e podem ser separados com **"&"** a cada novo parâmetro.

Eles são bem interessantes de se utilizar quando **entregamos um endpoint** que **pode ter vários parâmetros** para moldar a resposta, um bom exemplo é um **endpoint** que **entrega o resultado paginado**, podemos fazer com que a **resposta seja de acordo com os parâmetros escolhidos do cliente**:

```
@GetMapping("/my/{userId}")
public ResponseEntity<Page<SimplifiedGoalResponse>> find(@PathVariable Long userId,
    @RequestParam String sortBy,
    @RequestParam int size){
```

GET `http://localhost:8080/api/goals/my/1?sortBy=id&size=3`

```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

Request Line
Request Headers
Request Message Header
A blank line separates header & body
Request Message Body



RequestBody - Spring Web

Quando as **informações enviadas são extensas** ou **precisam ser fortemente protegidas**, elas são enviadas pelo **corpo da requisição**. Essas informações são conhecidas como **payload**, é muito mais complicado interceptar informações pelo payload da requisição do que pela url, além de possuir mais espaço para enviar objetos complexos:

```
@PostMapping("/{userId}")
public ResponseEntity<DetailedGoalResponse> createGoal(@PathVariable Long userId, @RequestBody @Valid
    @PostGoalRequest metaForm,
    @Autowired UriComponentsBuilder uriBuilder) {

    User user = userService.findById(userId);
    Goal goal = goalService.saveGoal(user, metaForm.convertToGoalModel());
    DetailedGoalResponse goalResponse = new DetailedGoalResponse(goal);

    URI uri = uriBuilder.path("/{api/goals/{id}}").buildAndExpand(goalResponse.getId()).toUri();
    return ResponseEntity.created(uri).body(goalResponse);
}
```

```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

Request Line
Request Headers
Request Message Header
A blank line separates header & body
Request Message Body



Validação e conversão de parâmetros - Spring Web

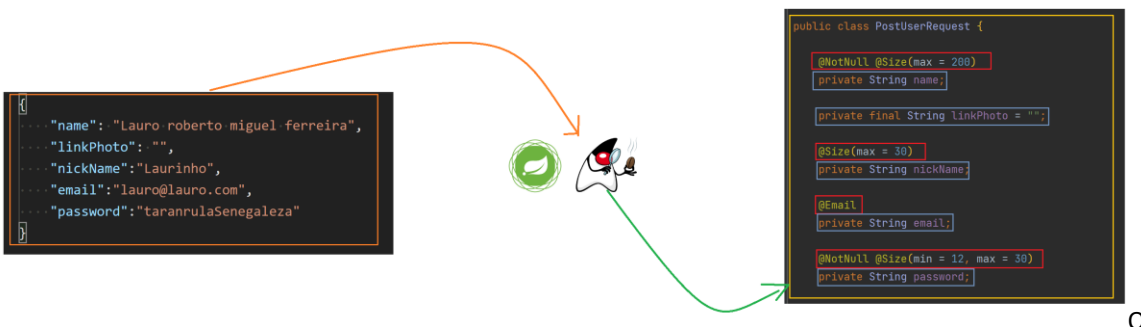
O **Spring** consegue automaticamente **converter** os **parâmetros** passados pelo cliente, tanto na **URI** como no **Body**, para os tipos específicos do Java, ou melhor, ele tenta fazer essa **conversão** e na maioria das vezes funciona:



Validação com Spring validation – Spring Web

A **conversão automática** para **variáveis de URL** do **Spring** serve muito bem, afinal de contas são poucos parâmetros e uma validação manual pode ser tranquila de se fazer. Mas quando queremos **pegar o valor de uma requisição pelo body** isso muda um pouco, geralmente estamos pegando um objeto complexo e com **vários atributos**.

Nesse tipo de cenário podemos usar o **Spring Validation**, com ele podemos anotar os **atributos** da **representação do body** e definir **restrições com anotações**, por exemplo, *tamanho, se é maior que 0, se não é branco ou nulo etc....*:



método do controller que vai usar essa validação precisa indicar isso com **@Valid** no parâmetro.

Outra dica para facilitar a conversão de determinados atributos é usar o **@JsonFormat** caso sua api trabalhe com Json, com essa **anotação** podemos especificar o formato esperado da data, **tornando fácil para o Spring** converter o atributo em uma **classe específica de Data do Java**:

