

9.1. Validação do modelo com Bean Validation

quinta-feira, 2 de março de 2023

20:30



Validação do modelo com Bean Validation

Quando se trabalha com entrada de dados de usuários, é importante fazer validações de tais dados, e ao construir uma API Rest com Java, podemos utilizar o **Bean Validation do JAVA EE**.

Especificação que faz parte do JEE e valida propriedades dos objetos Java. Trabalha com anotações para especificar os tipos de validações e especificar propriedades.

Podemos ter implementações do Bean Validation que possuem próprias validações customizadas e também **podemos especializar o Bean Validation para usar as próprias anotações de validações**.

Como o Bean Validation é uma especificação, temos que usar uma implementação, a mais conhecida é o Hibernate Validator e a dependência do Spring Boot traz consigo em suas dependências transitivas. O Hibernate Validator também traz a especificação como dependência, a Validation-api.

Edit: A partir da versão 2.3.0 do Spring o Bean Validation não é adicionado automaticamente como dependência do pacote spring-boot-starter-web.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

9.2. Adicionando constraints e validando no controller com @Valid

quinta-feira, 2 de março de 2023 20:40

Caso não haja anotação `@NotNull` o próprio banco de dados lança uma exceção para a adição de registro com campo nullo e é tratado pelo Spring

Cadeia de tratamento do erro:

```
java.sql.SQLIntegrityConstraintViolationException  
org.hibernate.exception.ConstraintViolationException  
org.springframework.dao.DataIntegrityViolationException
```

pois tentamos adicionar um registro com campo nullo no qual uma coluna é notnull

The screenshot shows a REST client interface. The top bar indicates a POST request to `http://localhost:8080/restaurantes`. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2   "taxaFrete": 12.00,  
3   "cozinha": {}  
4   "id": 1  
5 }  
6 }
```

Below the request, the response is shown with a status of 500 Internal Server Error. The response body is a JSON object:

```
1 {  
2   "status": 500,  
3   "type": "http://localhost:8080/erro-de-sistema",  
4   "title": "Erro de interno do sistema.",  
5   "detail": "Ocorreu um erro interno inesperado no sistema. Tente novamente e se o proplema persistir, entre em  
6     contato com o administrador do sistema.",  
7   "timeStamp": "2023-03-03T09:22:47.4115028"
```

Agora com `@NotNull` na propriedade na qual será validada temos um tratamento a nível do JEE pelo Bean Validation com a exceção `javax.validation.ConstraintViolationException` e trazendo uma lista de constraints violadas

```
javax.validation.ConstraintViolationException: Validation failed for classes [com.algaworks.algafood.domain.model.Restaurante] during persist time for groups [javax.validation.  
List of constraint violations:[  
  ConstraintViolationImpl{interpolatedMessage='não deve ser nulo', propertyPath=name, rootBeanClass=class com.algaworks.algafood.domain.model.Restaurante, messageTem  
] <22 internal lines>  
at org.springframework.orm.jpa.ExtendedEntityManagerCreator$ExtendedEntityManagerInvocationHandler.invoke(ExtendedEntityManagerCreator.java:362)
```

Mas podemos tratar entidades com propriedades anotadas com anotações do Bean Validation dentro da camada do controlador com `@Valid`, pois antes de fazer todo o processo de chamada de métodos nas camadas ele (Bean Validation com `@Valid`) faz as validações

```

@PostMapping
public ResponseEntity<> adicionar(@RequestBody @Valid Restaurante restaurante) {

    try {
        Restaurante restauranteSalvo = restauranteService.salvar(restaurante);
        return ResponseEntity.status(HttpStatus.CREATED).body(restauranteSalvo);
    } catch (CozinhaNaoEncontradaException e){
        throw new NegocioException(e.getMessage(), e);
    }
}

```

A exceção é capturada pelo nosso handle ApiExceptionHandler e tratada por MethodArgumentNotValidException pois de fato é um erro de sintaxe retornando um erro 400

The screenshot shows a REST client interface. At the top, a POST request is configured for the URL `http://localhost:8080/restaurantes`. The request body is a JSON object:

```

{
  "taxaFrete": 12.00,
  "cozinha": {
    "id": 1
  }
}

```

The response status is `400 Bad Request`. The response body is a JSON object:

```

{
  "status": 400,
  "title": "Bad Request",
  "userMessage": "Ocorreu um erro interno inesperado no sistema. Tente novamente e se o problema persistir, entre em contato com o administrador do sistema.",
  "timeStamp": "2023-03-03T10:40:02.6423349"
}

```

9.3. Desafio tratando exception de violação de constraints de validação

sexta-feira, 3 de março de 2023 09:25

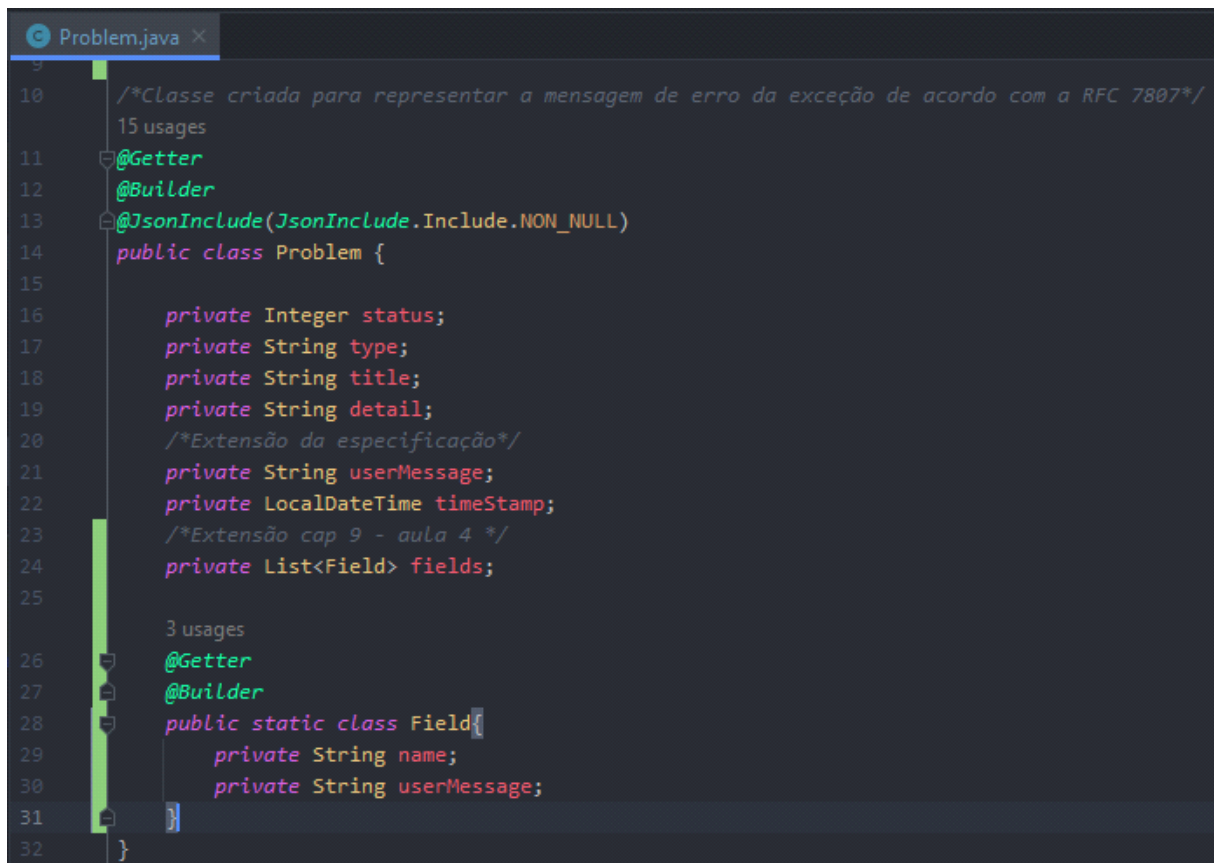
Ao validar uma classe com @Valid no controlador da nossa requisição, a exceção MethodArgumentNotValidException é lançada. O desafio é fazer esse tratamento e tornar a mensagem do erro mais legível e coerente. Como abaixo

```
{
  "status": 400,
  "timestamp": "2019-10-01T12:44:30.355301",
  "type": "https://algafood.com.br/dados-invalidos",
  "title": "Dados inválidos",
  "detail": "Um ou mais campos estão inválidos. Faça o preenchimento correto e tente novamente.",
  "userMessage": "Um ou mais campos estão inválidos. Faça o preenchimento correto e tente novamente."
}
```

9.4. Estendendo o Problem Details para adicionar as propriedades com constraints violadas

sexta-feira, 3 de março de 2023

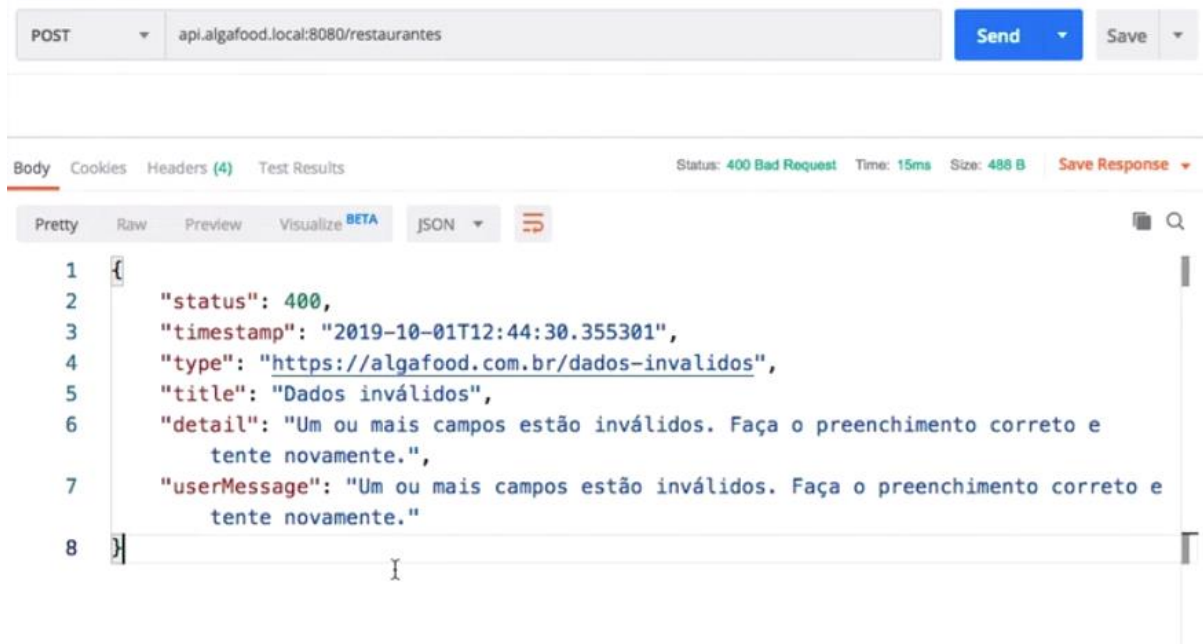
13:52



```
9
10  /*Classe criada para representar a mensagem de erro da exceção de acordo com a RFC 7807*/
11  15 usages
12  @Getter
13  @Builder
14  @JsonInclude(JsonInclude.Include.NON_NULL)
15  public class Problem {
16
17      private Integer status;
18      private String type;
19      private String title;
20      private String detail;
21      /*Extensão da especificação*/
22      private String userMessage;
23      private LocalDateTime timeStamp;
24      /*Extensão cap 9 - aula 4 */
25      private List<Field> fields;
26
27      3 usages
28      @Getter
29      @Builder
30      public static class Field{
31          private String name;
32          private String userMessage;
33      }
34  }
```

A ideia foi adicionar uma propriedade para a classe que representa o Problem Details ([8.17. Conhecendo a RFC 7807 \(Problem Details for HTTP APIs\)](#)) da aplicação.

A representação do erro antes:



Temos que adicionar uma propriedade que referencia os campos que estão com problemas, ou no caso, estamos passando null em propriedades não nulas no banco de dados.

```
Problem.java
9
10  /*Classe criada para representar a mensagem de erro da exceção de acordo com a RFC 7807*/
11  15 usages
12  @Getter
13  @Builder
14  @JsonInclude(JsonInclude.Include.NON_NULL)
15  public class Problem {
16
17      private Integer status;
18      private String type;
19      private String title;
20      private String detail;
21      /*Extensão da especificação*/
22      private String userMessage;
23      private LocalDateTime timeStamp;
24      /*Extensão cap 9 - aula 4 */
25      private List<Field> fields;
26
27      3 usages
28      @Getter
29      @Builder
30      public static class Field{
31          private String name;
32          private String userMessage;
33      }
34  }
```

vamos compor a propriedade com uma classe que encapsula o nome da propriedade e a mensagem da exceção violada.

```

@Override
protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex, HttpHeaders headers,
    HttpStatus status, WebRequest request) {

    final List<Problem.Field> problemFieldErrors = ex.getFieldErrors().stream()
        .map(fd -> Problem.Field.builder()
            .name(fd.getField())
            .userMessage(fd.getDefaultMessage())
            .build())
        .toList();

    final ProblemType problemType = ProblemType.DADOS_INVALIDOS;
    String detail = "Um ou mais campos estão invalidados. Faça o preenchimento e" +
        " tente novamente.";
    final Problem problema = createProblemType(status, problemType, detail)
        .userMessage(detail)
        .fields(problemFieldErrors)
        .build();
    return handleExceptionInternal(ex, problema, headers, status, request);
}

```

POST http://localhost:8080/restaurantes Send

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```

2 {
3   "nome": "teste",
4   "taxaFrete": 12.00,
5   "cozinha": "cozinha",
6   "id": 1
7 }

```

Body Cookies Headers (4) Test Results Status: 400 Bad Request Time: 11 ms Size: 576 B Save Response

Pretty Raw Preview Visualize JSON

```

1 {
2   "status": 400,
3   "type": "http://localhost:8080/dados-invalidos",
4   "title": "Dados inválidos",
5   "detail": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
6   "userMessage": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
7   "timeStamp": "2023-03-03T20:03:41.818106",
8   "fields": [
9     {
10      "name": "taxaFrete",
11      "userMessage": "não deve ser nulo"
12    },
13    {
14      "name": "nome",
15      "userMessage": "não deve ser nulo"
16    }
17  ]
18 }

```

Cookies
Capture requests
Runner
Trash

9.5. Conhecendo e adicionando mais constraints de validação no modelo

sexta-feira, 3 de março de 2023 20:47

@NotNull

Verifica se o valor anotado não é null

@NotEmpty

Verifica se o elemento anotado não é nulo nem vazio

@NotBlank

Verifica se a sequência de caracteres anotada não é nula e se o comprimento aparado é maior que 0. A diferença para @NotEmpty é que essa restrição só pode ser aplicada em sequências de caracteres e os espaços em branco à direita são ignorados.

@PositiveOrZero

Verifica se o elemento é positivo ou zero.

@DecimalMax(value=, inclusive=)

Verifica se o valor anotado é menor que o máximo especificado, quando inclusive=false. Caso contrário, se o valor for menor ou igual ao máximo especificado. O valor do parâmetro é a representação de string do valor máximo de acordo com a representação de string BigDecimal.

AlgaFood / Restaurante / Restaurante - adicionar

POST http://localhost:8080/restaurantes

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   ...
3   "taxaFrete": -12.00,
4   "cozinha": {
```

Body Cookies Headers (4) Test Results Status: 400 Bad Request Time: 9 ms Size: 593 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "status": 400,
3   "type": "http://localhost:8080/dados-invalidos",
4   "title": "Dados inválidos",
5   "detail": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
6   "userMessage": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
7   "timestamp": "2023-03-03T22:48:02.2561921",
8   "fields": [
9     {
10      "name": "nome",
11      "userMessage": "não deve estar em branco"
12    },
13    {
14      "name": "taxaFrete",
15      "userMessage": "deve ser maior ou igual a 0"
16    }
17  ]
18 }
```

https://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/#section-builtin-constraints

9.6. Validando as associações de uma entidade em cascata

sexta-feira, 3 de março de 2023

22:48

Ao tentarmos adicionar uma entidade cuja uma das propriedades for uma entidade com propriedades nulas, o erro

`org.springframework.dao.InvalidDataAccessApiUsageException: O id fornecido não pode ser nulo!` será lançado

```
1  {  
2    "nome": "Chilena",  
3    "taxaFrete": 12.00,  
4    "cozinha": {  
5      "id": null  
6    }  
7  }
```

Podemos adicionar uma validação na propriedade da entidade

```
@ManyToOne(fetch = FetchType.LAZY)  
@NotNull  
@JsonIgnoreProperties("hibernateLazyInitializer")  
@JoinColumn(name = "cozinha_id", nullable = false)  
private Cozinha cozinha;
```

mas o id não será validado, apenas a propriedade como um todo

```
@NotNull  
@EqualsAndHashCode.Include  
@Id  
@GeneratedValue(strategy = GenerationType.IDENTITY)//provedor de persistencia  
private Long id;
```

ao adicionar uma validação no ID da entidade da propriedade, também não será validado, pois o Bean Validation não validação em cascata, tempos que explicitar isso na definição da propriedade com `@Valid`. Logo, o Bean Validation não irá apenas validar a entidade mas também suas propriedades.

POST http://localhost:8080/restaurantes Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "nome": "Chilena",
3   "taxaFrete": 12.00,
4   "cozinha": {
5     "id": null
6   }
7 }
```

Body Cookies Headers (4) Test Results Status: 400 Bad Request Time: 33 ms Size: 527 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "status": 400,
3   "type": "http://localhost:8080/dados-invalidos",
4   "title": "Dados inválidos",
5   "detail": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
6   "userMessage": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
7   "timestamp": "2023-03-04T00:08:21.7322364",
8   "fields": [
9     {
10      "name": "cozinha.id",
11      "userMessage": "não deve ser nulo"
12    }
13  ]
14 }
```

9.7. Agrupando e restringindo constraints que devem ser usadas na validação

sábado, 4 de março de 2023 00:09

The screenshot shows a REST client interface with a POST request to `http://localhost:8080/restaurantes`. The request body is a JSON object:

```
{
  "nome": "Chilena",
  "taxaFrete": 12.00,
  "cozinha": 4,
  "id": 2
}
```

The response is a 400 Bad Request with the following JSON body:

```
{
  "status": 400,
  "type": "http://localhost:8080/dados-invalidos",
  "title": "Dados inválidos",
  "detail": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
  "userMessage": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
  "timeStamp": "2023-03-04T11:18:20.8899247",
  "fields": [
    {
      "name": "cozinha.nome",
      "userMessage": "não deve ser nulo"
    }
  ]
}
```

Ao adicionarmos um Restaurante que contém uma Cozinha, devemos passar apenas o id de Cozinha, mas a validação ocorre em outras propriedades.

The screenshot shows a REST client interface with a POST request to `http://localhost:8080/cozinhas`. The request body is a JSON object:

```
{
  "nome": "Tailandesa"
}
```

The response is a 400 Bad Request with the following JSON body:

```
{
  "status": 400,
  "type": "http://localhost:8080/dados-invalidos",
  "title": "Dados inválidos",
  "detail": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
  "userMessage": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
  "timeStamp": "2023-03-04T11:21:33.1184242",
  "fields": [
    {
      "name": "id",
      "userMessage": "não deve ser nulo"
    }
  ]
}
```

No caso de cadastro de Cozinha que não precisamos passar o Id, a validação ocorre também.

Podemos alterar esse comportamento aplicando grupos para as validações

```
package com.algaworks.algafood;

public interface Groups {

    public interface Restaurante{}

}
```

Podemos criar uma classe ou interface que e dentro podemos definir classes/interfaces para servir de itens do agrupamento. No caso acima, validadores para cadastro de restaurante

```
@PostMapping
public ResponseEntity<?> adicionar
    (@RequestBody @Validated(Groups.CadastroRestaurante.class) Restaurante restaurante) {

    try {
        Restaurante restauranteSalvo = restauranteService.salvar(restaurante);
        return ResponseEntity.status(HttpStatus.CREATED).body(restauranteSalvo);
    } catch (CozinhaNaoEncontradaException e){
        throw new NegocioException(e.getMessage(), e);
    }
}
```

Em vez de utilizarmos @Valid, utilizaremos @Validated passando como propriedade a classe para fazer o agrupamento.

Uma requisição POST de adição de um Restaurante requer o campo nome, taxaFrete, Cozinha e cozinha.id não nulos. Tempo que aplicar o grupo a qual as validações dessas propriedades pertencem

```

25
26     @EqualsAndHashCode.Include
27     @Id
28     @GeneratedValue(strategy = GenerationType.IDENTITY) //provedor de persistencia
29     private Long id;
30
31     @NotBlank(groups = Groups.CadastroRestaurante.class)
32     @Column(nullable = false)
33     private String nome;
34
35     @PositiveOrZero(groups = Groups.CadastroRestaurante.class)
36     @Column(name = "taxa_frete", nullable = false)
37     private BigDecimal taxaFrete;
38
39     @NotNull(groups = Groups.CadastroRestaurante.class)
40     @Valid
41     @ManyToOne(fetch = FetchType.LAZY)
42     @JsonIgnoreProperties("hibernateLazyInitializer")
43     @JoinColumn(name = "cozinha_id", nullable = false)
44     private Cozinha cozinha;

```

na entidade Restaurante, serão validados somente as validações na qual estão declaradas no grupo da anotação @Validated. Na entidade Cozinha, é necessário declarar @valid para validar as anotações da propriedade.

```

10     import java.util.ArrayList;
11     import java.util.List;
12
13     29 usages
14     @Data
15     @EqualsAndHashCode(doNotUseGetters = true)
16     @Entity
17     @Table(name = "cozinha")
18     public class Cozinha {
19
20         @NotNull(groups = Groups.CadastroRestaurante.class)
21         @EqualsAndHashCode.Include
22         @Id
23         @GeneratedValue(strategy = GenerationType.IDENTITY)//provedor de persistencia
24         private Long id;
25
26         @NotNull
27         @Column(name = "nome", length = 30, nullable = false)
28         private String nome;
29

```

em Cozinha dentro de Restaurante, será anotado com o grupo somente a propriedade Id, que é necessário para a requisição.

POST http://localhost:8080/restaurantes

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   ...
3   "taxaFrete": 12.00,
4   "cozinha": {
5     "id": null
6   }
7 }
```

Body Cookies Headers (4) Test Results Status: 400 Bad Request Time: 107 ms Size: 585 B Save Response

Pretty Raw Preview Visualize JSON

```
3 "type": "http://localhost:8080/dados-invalidos",
4 "title": "Dados inválidos",
5 "detail": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
6 "userMessage": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
7 "timestamp": "2023-03-04T12:09:53.8135121",
8 "fields": [
9   {
10    "name": "nome",
11    "userMessage": "não deve estar em branco"
12  },
13  {
14    "name": "cozinha.id",
15    "userMessage": "não deve ser nulo"
16  }
17 ]
```

e no cadastro de Cozinha que é necessário apenas Nome

POST http://localhost:8080/cozinhas

Params Authorization Headers (5) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "nome": "Tailandesa"
3 }
```

Body Cookies Headers (5) Test Results Status: 201 Created Time: 79 ms Size: 197 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 5,
3   "nome": "Tailandesa"
4 }
```

Pois quando não especificamos nenhum grupo no @Valid, ela entra no grupo Default, no qual todas as validações da propriedade entram no grupo também.

9.8. Convertendo grupos de constraints para validação em cascata com @ConvertGroup

sábado, 4 de março de 2023 13:35

Embora as anotações nas propriedades definindo os grupos funcione, com o tempo fica muito poluído e cada propriedade tem que ser anotada. Para contornar isso podemos converter o grupo Default de uma propriedade para outro grupo.

```
@PostMapping
public ResponseEntity<?> adicionar
    (@RequestBody @Valid Restaurante restaurante) {

    try {
        Restaurante restauranteSalvo = restauranteService.salvar(restaurante);
        return ResponseEntity.status(HttpStatus.CREATED).body(restauranteSalvo);
    } catch (CozinhaNaoEncontradaException e) {
        throw new NegocioException(e.getMessage(), e);
    }
}
```

Definimos somente @Valid para a entidade no controlador que por padrão o grupo Default é definido nas entidade.

```
@ConvertGroup(to = Groups.CozinhaId.class)
@NotNull
@Valid
@ManyToOne
@JoinColumn(name = "cozinha_id", nullable = false)
private Cozinha cozinha;
```

na entidade Cozinha converteremos o grupo das validações

```
29 usages
@Data
@EqualsAndHashCode(doNotUseGetters = true)
@Entity
@Table(name = "cozinha")
public class Cozinha {

    @NotNull(groups = Groups.CozinhaId.class)
    @EqualsAndHashCode.Include
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)//provedor de persistencia
    private Long id;

    @NotNull
    @Column(name = "nome", length = 30, nullable = false)
    private String nome;

    @JsonIgnore
    @OneToMany(mappedBy = "cozinha")
    private List<Restaurante> restaurantes = new ArrayList<>();
}
```

9.9. Desafio adicionando constraints de validação no modelo

sábado, 4 de março de 2023

13:39

```
package com.algaworks.algafood;

public interface Groups {

    2 usages
    public interface CozinhaId {}
    2 usages
    public interface EstadoId {}
}
```

```
@PostMapping
public ResponseEntity<?> adicionar(@RequestBody @Valid Cidade cidade) {
    try {
        Cidade c = cidadeService.salvar(cidade);
        return ResponseEntity.status(HttpStatus.CREATED).body(c);
    } catch (EstadoNaoEncontradoException e) {
        throw new NegocioException(e.getMessage(), e);
    }
}
```

```
@ConvertGroup(to = Groups.EstadoId.class)
@ManyToOne
@NotNull
@Valid
@JoinColumn(nullable = false)
private Estado estado;
```


9.10. Customizando mensagens de validação na anotação da constraint

sábado, 4 de março de 2023

14:10

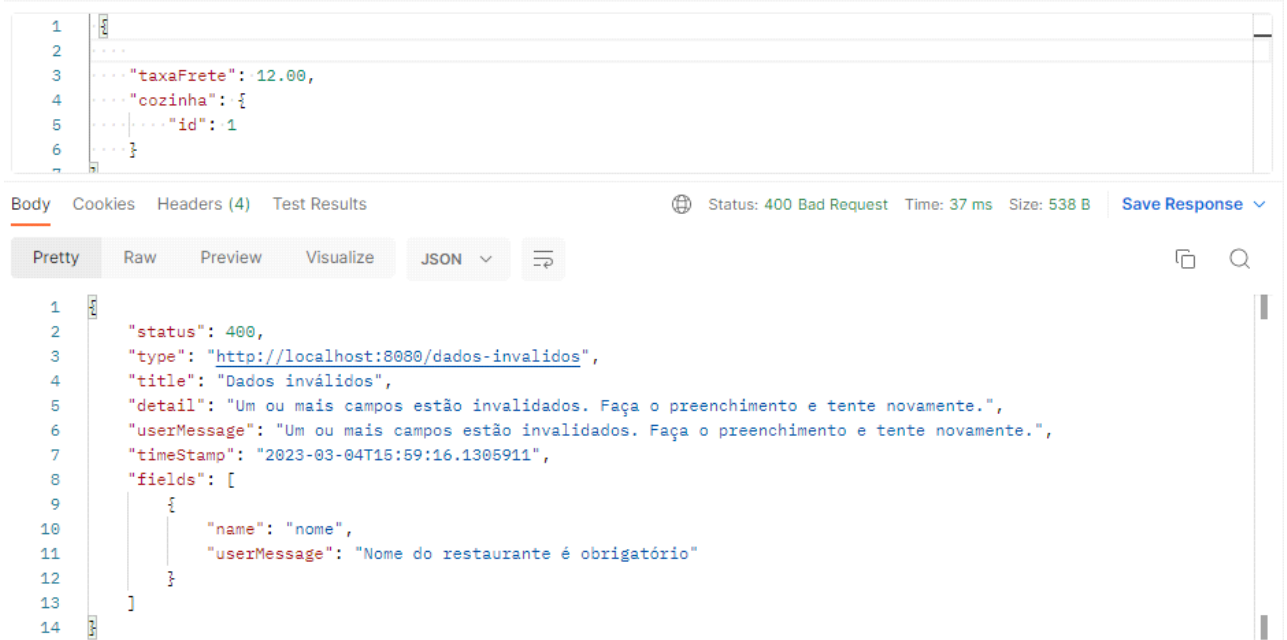
As constraints aceitam mensagens personalizadas

```
@NotEmpty(message = "Não é permitido um campo vazio")  
@Column(nullable = false)  
private String nome;
```

9.11. Customizando e resolvendo mensagens de validação globais em Resource Bundle

sábado, 4 de março de 2023 14:50

Podemos customizar as mensagens na resposta do erro em uma validação de propriedades



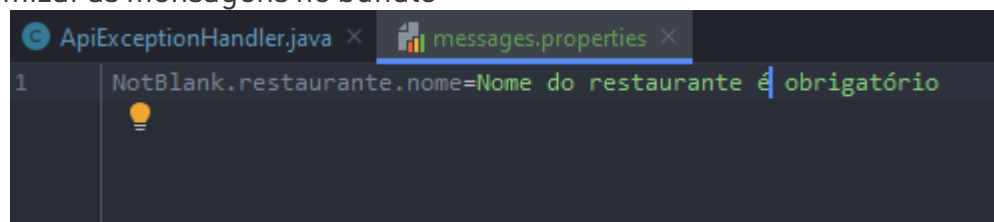
```
1 {
2   ...
3   ... "taxaFrete": 12.00,
4   ... "cozinha": {
5     ... "id": 1
6   }
7 }
8
9
10
11
12
13
14
```

Body Cookies Headers (4) Test Results Status: 400 Bad Request Time: 37 ms Size: 538 B Save Response

Pretty Raw Preview Visualize JSON

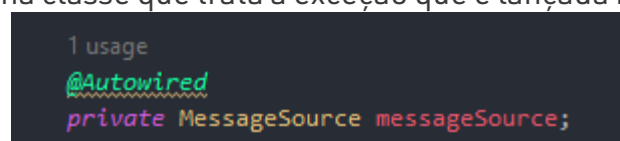
```
1 {
2   "status": 400,
3   "type": "http://localhost:8080/dados-invalidos",
4   "title": "Dados inválidos",
5   "detail": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
6   "userMessage": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
7   "timestamp": "2023-03-04T15:59:16.1305911",
8   "fields": [
9     {
10      "name": "nome",
11      "userMessage": "Nome do restaurante é obrigatório"
12    }
13  ]
14 }
```

Primeiro temos que criar um arquivo bundle (Resource Bundle). Para cada entidade temos que customizar as mensagens no bundle



```
1NotBlank.restaurante.nome=Nome do restaurante é obrigatório
```

e injetar a interface na classe que trata a exceção que é lançada `MethodArgumentNotValid`



```
1 usage
2 @Autowired
3 private MessageSource messageSource;
```

dentro do método que trata a exceção , podemos fazer a seguinte modificação

de:

```

@Override
protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex, HttpHeaders headers,
                                                                HttpStatus status, WebRequest request) {

    final List<Problem.Field> problemFieldErrors = ex.getFieldErrors().stream()
        .map(fd -> Problem.Field.builder()
            .name(fd.getField())
            .userMessage(fd.getDefaultMessage())
            .build())
        .toList();

    final ProblemType problemType = ProblemType.DADOS_INVALIDOS;
    String detail = "Um ou mais campos estão invalidados. Faça o preenchimento e" +
        " tente novamente.";
    final Problem problema = createProblemType(status, problemType, detail)
        .userMessage(detail)
        .fields(problemFieldErrors)
        .build();
    return handleExceptionInternal(ex, problema, headers, status, request);
}

```

para:

```

@Override
protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex, HttpHeaders headers,
                                                                HttpStatus status, WebRequest request) {

    final List<Problem.Field> problemFieldErrors = ex.getFieldErrors().stream()
        .map(fd -> {
            String message = messageSource.getMessage(fd, LocaleContextHolder.getLocale());
            return Problem.Field.builder()
                .name(fd.getField())
                .userMessage(message)
                .build();
        })
        .toList();
}

```

messageSource.getMessage aceita no 1º argumento um MessageSourceResolvable

```

.stream()
.map(fd -> {
    String message = messageSource.getMessage(fd, LocaleContextHolder.getLocale());
    return Problem.Field.builder()
        .name(fd.getField())
        .userMessage(message)
        .build();
})
.toList();

```

Um FieldError é desse tipo e é aceito como argumento, e outro é um Locale.

Podemos customizar as mensagens do bundle de uma forma mais abrangente ou individual

```
ApiExceptionHandler.java x messages.properties x Cozinha.java x
1 #Podemos customizar individualmente cada entida e e atributo
2
3 #NotBlank.restaurante.nome=Nome do restaurante é obrigatório
4 #NotBlank.Cozinha.nome=Nome do restaurante é obrigatório
5
6 #ou utilizando uma mensagem mais genérica e abrangente como:
7 #NotBlank=é obrigatório
8
9 #Podemos customizar a mensagem generica com PlaceHolders entre chaves
10 NotBlank={0} é obrigatório
11
12 #e substituir a variavel entre chaves, setando os valores
13 #nome=nome é obrigatório
14 restaurante.nome=Nome do restaurante
15 cozinha.nome=Nome da cozinha
```

POST http://localhost:8080/restaurantes Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   ...
3   "taxaFrete": 12.00,
4   "cozinha": {
5     "id": 1
6   }
7 }
```

Body Cookies Headers (4) Test Results Status: 400 Bad Request Time: 31 ms Size: 538 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "status": 400,
3   "type": "http://localhost:8080/dados-invalidos",
4   "title": "Dados inválidos",
5   "detail": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
6   "userMessage": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
7   "timeStamp": "2023-03-04T18:22:40.4554023",
8   "fields": [
9     {
10      "name": "nome",
11      "userMessage": "Nome do restaurante é obrigatório"
12    }
13  ]
14 }
```

POST http://localhost:8080/cozinhas

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "nome" : null
3 }
```

Body Cookies Headers (4) Test Results Status: 400 Bad Request Time: 79 ms Size: 534 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "status": 400,
3   "type": "http://localhost:8080/dados-invalidos",
4   "title": "Dados inválidos",
5   "detail": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
6   "userMessage": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
7   "timeStamp": "2023-03-04T18:22:34.6255438",
8   "fields": [
9     {
10      "name": "nome",
11      "userMessage": "Nome da cozinha é obrigatório"
12    }
13  ]
14 }
```

9.12. Desafio customizando mensagens de validação

sábado, 4 de março de 2023

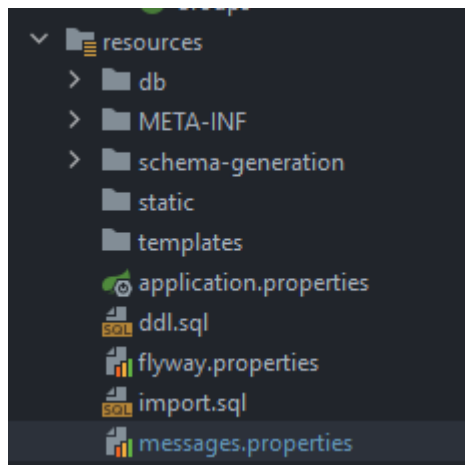
18:24

```
messages.properties x
5
6 #ou utilizando uma mensagem mais genérica e abrangente como:
7 #NotBlank=é obrigatório
8
9 #Podemos customizar a mensagem generica com PLaceHolders entre chaves
10 NotBlank={0} é obrigatório
11 NotNull={0} é obrigatório
12 PositiveOrZero={0} deve ser um valor maior ou igual a zero
13
14 #e substituir a variavel entre chaves, setando os valores
15 #nome=nome é obrigatório
16
17 #9.12. Desafio: customizando mensagens de validação
18
19 # Cozinha
20 cozinha.nome=Nome da cozinha
21 cozinha.id=Código da cozinha
22
23 # Restaurante
24 NotNull.restaurante.taxaFrete={0} é obrigatória
25 NotNull.restaurante.cozinha={0} é obrigatória
26 restaurante.nome=Nome do restaurante
27 restaurante.cozinha=Cozinha do restaurante
28 restaurante.taxaFrete=Taxa de frete do restaurante
29
30 # Estado
31 estado.nome=Nome do estado
32 estado.id=Código do estado
33
34 # Cidade
35 cidade.nome=Nome da cidade
36 cidade.estado=Estado da cidade
```

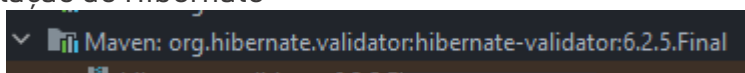
9.13. Resolvendo mensagens de validação com Resource Bundle do Bean Validation

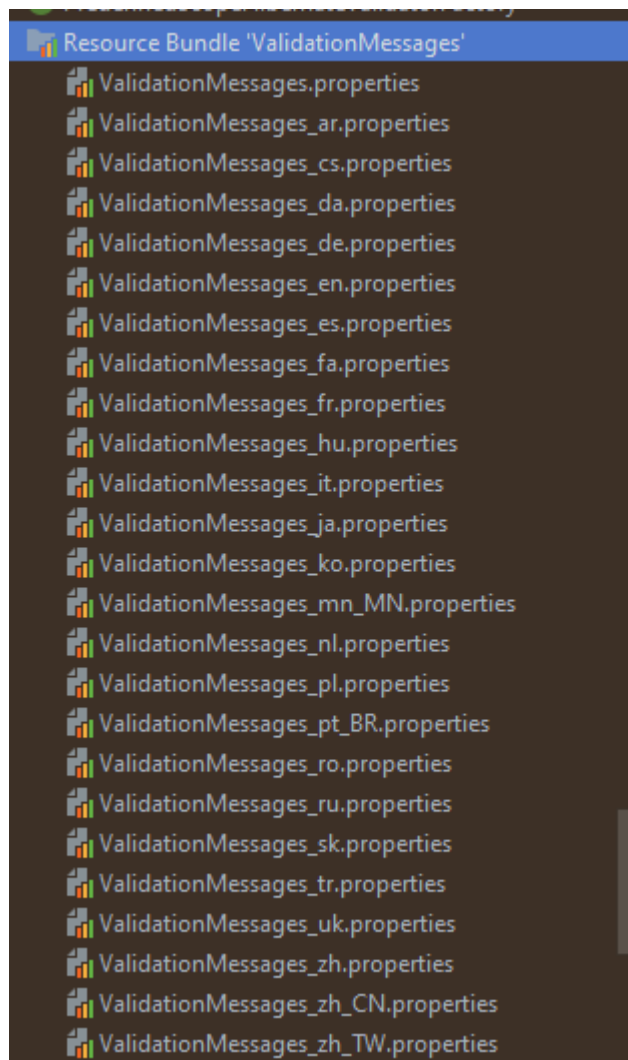
sábado, 4 de março de 2023 19:08

Quanto utilizamos o resource bundle na pasta src/main/resources

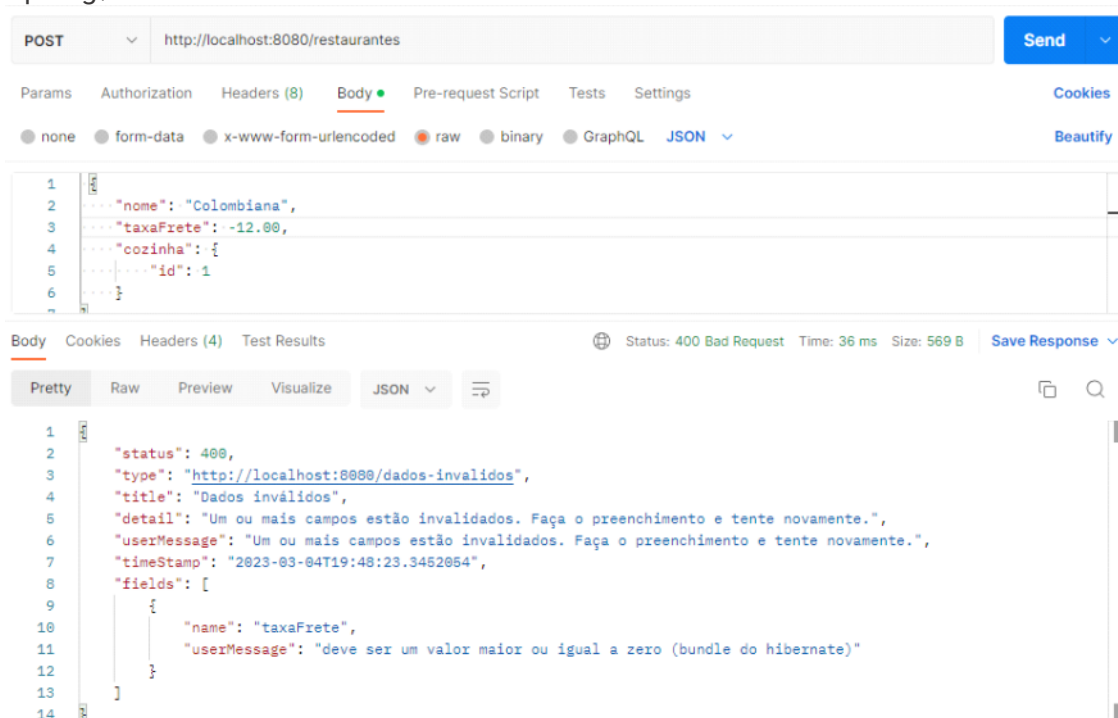


estamos na verdade utilizando um resource bundle gerenciado pelo Spring. Mas temos o da implementação do Hibernate

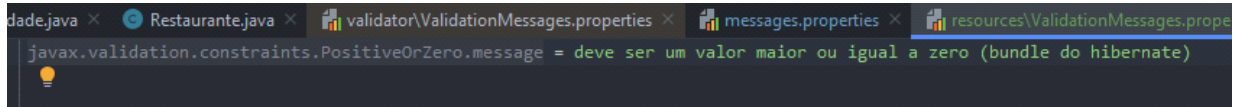
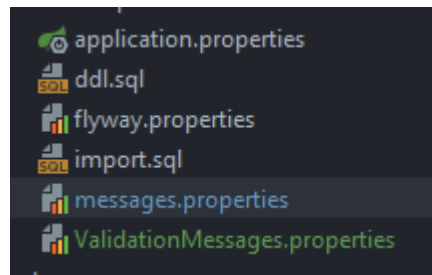




e podemos customizar essas mensagens da implementação (para caso não usar Spring)



Temos que criar um arquivo exatamente do mesmo nome que o bundle original do Hibernate e adicionar a propriedade da anotação



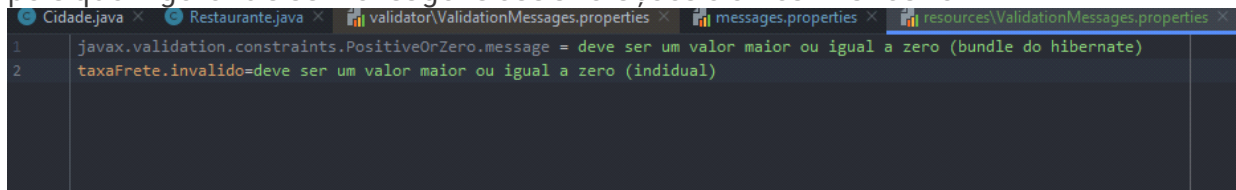
e descomentar a mensagem customizada de messages.properties

```
#Podemos customizar a mensagem generica com PlaceHolders entre c
NotBlank={0} é obrigatório
NotNull={0} é obrigatório
#PositiveOrZero={0} deve ser um valor maior ou igual a zero
```

Se quisermos criar uma mensagem para uma propriedade específica, temos que modificar o resource bundle da implementação e criar nossa própria propriedade, como:

```
@NotNull
@PositiveOrZero(message = "{taxaFrete.invalido}")
@Column(name = "taxa_frete", nullable = false)
private BigDecimal taxaFrete;
```

agora precisamos criar uma propriedade dentro de ValidationMessages.properties pois quem gerencia as mensagens das anotações é o Bean Validation.



POST ▼ http://localhost:8080/restaurantes Send ▼

Params Authorization Headers (8) **Body** ● Pre-request Script Tests Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼ Beautify

```
1 {
2   "nome": "Colombiana",
3   "taxaFrete": -12.00,
4   "cozinha": {
5     "id": 1
6   }
7 }
```

Body Cookies Headers (4) Test Results 🌐 Status: 400 Bad Request Time: 67 ms Size: 558 B Save Response ▼

Pretty Raw Preview Visualize **JSON** ▼ 🔍

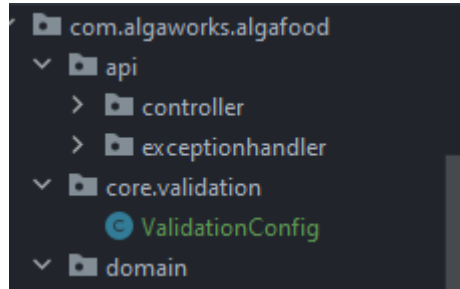
```
1 {
2   "status": 400,
3   "type": "http://localhost:8080/dados-invalidos",
4   "title": "Dados inválidos",
5   "detail": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
6   "userMessage": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
7   "timeStamp": "2023-03-04T21:02:44.2842803",
8   "fields": [
9     {
10      "name": "taxaFrete",
11      "userMessage": "deve ser um valor maior ou igual a zero (individual)"
12    }
13  ]
14 }
```

🍪 Cookies 📄 Captura requête 🗑️ Runar 🗑️ Trash 🔍

9.14. Usando o Resource Bundle do Spring como Resource Bundle do Bean Validation

sábado, 4 de março de 2023 21:03

Na aula configuramos para que o nosso projeto use apenas 1 resource bundle (e não 2 como na aula anterior)



o pacote core.validation foi criado pois não se encaixa em configurações de api, de domínio ou infrastructure

```
@Configuration
public class ValidationConfig {

    @Bean
    public LocalValidatorFactoryBean validator(MessageSource messageSource){
        LocalValidatorFactoryBean bean = new LocalValidatorFactoryBean();
        bean.setValidationMessageSource(messageSource);
        return bean;
    }
}
```

```
Cidade.java x Restaurante.java x messages.properties x ValidationConfig.java x
3 #Ou utilizando uma mensagem mais genérica e abrangente como:
4
5 #ou utilizando uma mensagem mais genérica e abrangente como:
6 #NotBlank={0} é obrigatório
7
8 #Podemos customizar a mensagem generica com PlaceHolders entre chaves
9 NotBlank={0} é obrigatório
10 NotNull={0} é obrigatório
11 #PositiveOrZero={0} deve ser um valor maior ou igual a zero
12
13 #e substituir a variavel entre chaves, setando os valores
14 #nome=nome é obrigatório
15
16 #9.12. Desafio: customizando mensagens de validação
17
18 # Cozinha
19 cozinha.nome=Nome da cozinha
20 cozinha.id=Código da cozinha
21
22 # Restaurante
23 NotNull.restaurante.taxaFrete={0} é obrigatória
24 NotNull.restaurante.cozinha={0} é obrigatória
25 restaurante.nome=Nome do restaurante
26 restaurante.cozinha=Cozinha do restaurante
27 restaurante.taxaFrete=Taxa de frete do restaurante
28
29 # Estado
30 estado.nome=Nome do estado
31 estado.id=Código do estado
32
33 # Cidade
34 cidade.nome=Nome da cidade
35 cidade.estado=Estado da cidade
36
37 #javax.validation.constraints.PositiveOrZero.message = deve ser um valor maior ou igual a zero (bundle do hibernate)
38 #taxaFrete.invalido=deve ser um valor maior ou igual a zero (individual)
```

Agora temos somente 1 resource bundle

9.15. Criando constraints de validação customizadas usando composição

sábado, 4 de março de 2023 21:16

Vamos criar uma anotação que significa uma validação

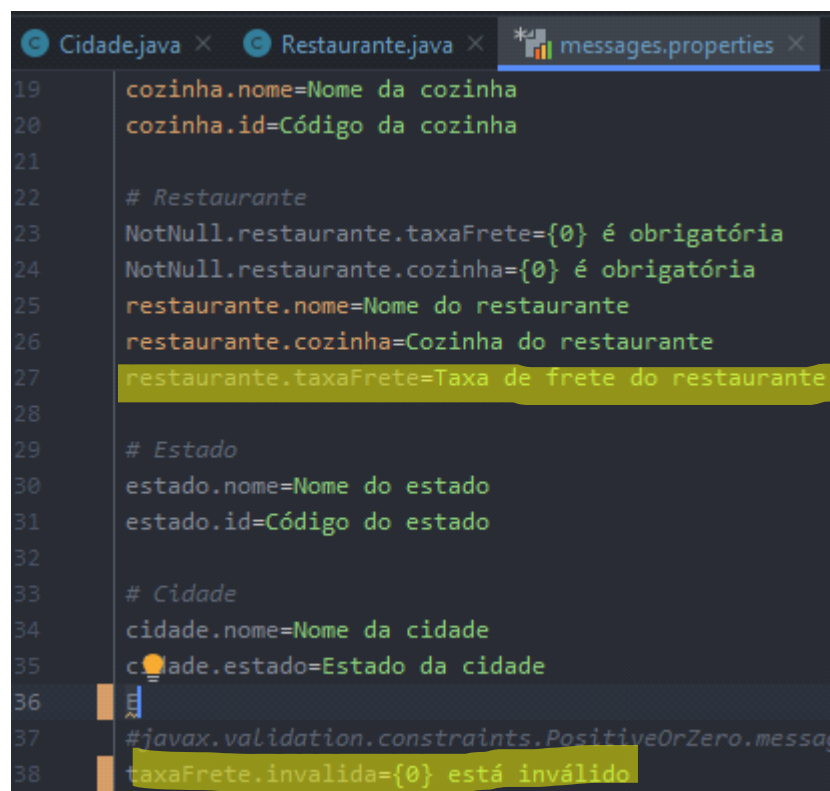
```
2 usages
@Target({ElementType.METHOD, ElementType.FIELD, ElementType.ANNOTATION_TYPE, ElementType.CONSTRUCTOR,
        ElementType.PARAMETER, ElementType.TYPE_USE})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = {})
@PositiveOrZero
public @interface TaxaFrete {

    @OverrideAttribute(constraint = PositiveOrZero.class, name = "message")
    String message() default "{taxaFrete.invalida}";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

}
```



The screenshot shows an IDE with three tabs: Cidade.java, Restaurante.java, and messages.properties. The messages.properties file contains the following content:

```
19 cozinha.nome=Nome da cozinha
20 cozinha.id=Código da cozinha
21
22 # Restaurante
23 NotNull.restaurante.taxaFrete={0} é obrigatória
24 NotNull.restaurante.cozinha={0} é obrigatória
25 restaurante.nome=Nome do restaurante
26 restaurante.cozinha=Cozinha do restaurante
27 restaurante.taxaFrete=Taxa de frete do restaurante
28
29 # Estado
30 estado.nome=Nome do estado
31 estado.id=Código do estado
32
33 # Cidade
34 cidade.nome=Nome da cidade
35 cidade.estado=Estado da cidade
36
37 #javax.validation.constraints.PositiveOrZero.message
38 taxaFrete.invalida={0} está inválido
```

The screenshot shows a REST client interface. The top bar indicates a POST request to `http://localhost:8080/restaurantes`. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2   "nome": "Colombiana",
3   "taxaFrete": -12.00,
4   "cozinha": {
5     "id": 1
6   }
7 }
```

Below the request, the response is shown with a status of 400 Bad Request. The response body is a JSON object:

```
1 {
2   "status": 400,
3   "type": "http://localhost:8080/dados-invalidos",
4   "title": "Dados inválidos",
5   "detail": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
6   "userMessage": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
7   "timeStamp": "2023-03-04T22:57:30.8306674",
8   "fields": [
9     {
10      "name": "taxaFrete",
11      "userMessage": "Taxa de frete do restaurante está inválido"
12    }
13  ]
14 }
```

Carlos Eduardo Guerra Resende TUTOR

30/06/2022 às 20:54

Olá CLAUDINEI MACHADO, boa noite!

Tudo bem amigo? Claro! Vamos buscar explicar cada linha destas para que o entendimento fique mais claro, ok? Vamos lá:

`String message() default "descrição obrigatória inválida";`

Com esta declaração definimos um mensagem default para o atributo message da nossa anotação. Isso significa dizer que se não definirmos o message ao utilizar a nossa anotação, esta mensagem será exibida em caso de falha na validação da anotação.

`Class<?>[] groups() default {};`

Com esta declaração habilitamos a possibilidade de definir grupos de validação para os quais a anotação será validada. Exemplo: você pode definir um grupo chamado `GroupUpdate`, `InsertUpdate` e `deleteUpdate` e para a sua anotação informar que somente em casos de `Update` e `Insert`, a anotação seria validada.

`Class<? extends Payload>[] payload() default {};`

Em relação a esta declaração estamos definindo um grupo de diferentes níveis de validação para uma anotação. Imagine que você deseje indicar o nível de severidade da falha de validação da anotação. Por exemplo, `erro`, `warning`, `info`, etc. Você poderia então definir diferentes níveis de severidade e utilizar para o atributo `payload` de sua anotação.

`String valueField();`

`String descriptionField();`

`String descriptionRequired();`

Em relação a estas declarações, você está definindo os seus próprios atributos da anotação, para os quais você fará as sua regras de validação na implementação da anotação.

Espero ter ajudado com o entendimento. Me deixe saber se há algo mais que

possamos ajudar a esclarecer. Um grande abraço e excelentes estudos.

9.16. Criando constraints de validação customizadas com implementação de ConstraintValidator

domingo, 5 de março de 2023

10:12

Nossa anotação

```
4 usages
@Target({ElementType.METHOD, ElementType.FIELD, ElementType.ANNOTATION_TYPE, ElementType.CONSTRUCTOR,
        ElementType.PARAMETER, ElementType.TYPE_USE})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = {MultiploValidator.class})
public @interface Multiplo {
    String message() default "TaxaFrete.multiplo";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

    2 usages
    int numero();
}
```

lógica do validador para numeros múltiplos de 5

```
1 usage
7 public class MultiploValidator implements ConstraintValidator<Multiplo, Number> {
8
9     /*Valor que é passado em TaxaFrete*/
10    2 usages
11    int numero;
12    @Override
13    public void initialize(Multiplo constraintAnnotation) {
14        this.numero = constraintAnnotation.numero();
15    }
}
```



```

16      @Override
17      public boolean isValid(Number number,
18                             ConstraintValidatorContext constraintValidatorContext) {
19
20          boolean isValid = true;
21
22          if(number != null){
23              /*valorDecimal = valor de taxaFrete do Restaurante em Decimal*
24               valorMultiplo = valor do múltiplo definido na propriedade da
25               declaração da anotação*/
26              var valorDecimal :BigDecimal = BigDecimal.valueOf(number.longValue());
27              var valorMultiplo :BigDecimal = BigDecimal.valueOf((numero));
28
29              /*Para verificar se é múltiplo:
30               * se valorDecimal dividido por valorMultiplo for igual a zero
31               * retorna true = é múltiplo
32               * retorna false = não é múltiplo*/
33              final BigDecimal remainder = valorDecimal.remainder(valorMultiplo);
34              isValid = 0 == remainder.compareTo(BigDecimal.ZERO);
35          }
36          return isValid;
37      }
38  }

```

Mensagens:

```

8      #Podemos customizar a mensagem generica com PlaceHolders entre chaves
9      NotBlank={0} é obrigatório
10     NotNull={0} é obrigatório
11     Multiplo={0} deve ser um valor multiplo de {1}
12     #PositiveOrZero={0} deve ser um valor maior ou igual a zero
13

```

```

# Restaurante
NotNull.restaurante.taxaFrete={0} é obrigatória
NotNull.restaurante.cozinha={0} é obrigatória
restaurante.nome=Nome do restaurante
restaurante.cozinha=Cozinha do restaurante
restaurante.taxaFrete=Taxa de frete do restaurante

```

Ficará = Taxa de frete do restaurante deve ser um valor múltiplo de {valor}

9.17. Criando constraints de validação customizadas em nível de classe

domingo, 5 de março de 2023 11:15

Podemos criar esse tipo de validação para criar validações condicionais nas propriedades, checando a partir de 1 propriedade para validar a próxima.

Quando temos esse tipo de necessidade complexa, podemos usar esse tipo.

Problema:

Para todo restaurante com frete grátis ($\text{taxaFrete} == 0$), o nome não deve ser nulo e deve conter "Frete Grátis" na descrição ou lança uma exceção, se não conter frete grátis ($\text{taxaFrete} > 0$) também não lança uma exceção.

primeiro vamos anotar a classe e depois fazer a anotation

```
@ValorZeroIncluiDescricao(valorField = "taxaFrete", descricaoField = "nome", descricaoObrigatoria="Frete Grátis")
@Data
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
@Entity
public class Restaurante {
```

A anotação será gerenciada pelo Bean Validation com o nome `@ValorZeroIncluiDescricao` e tem as seguintes propriedades:

- `valorField = "taxaFrete"`: o campo `valorField` terá o nome do atributo da taxa de frete do objeto do tipo `Restaurante`
- `descricaoField="nome"`: o campo `descricaoField` terá o nome do atributo nome do objeto do tipo `Restaurante`
- `descricaoObrigatoria="Frete Grátis"`: o campo `descricaoObrigatoria` terá como valor a string "Frete Grátis".

As 3 propriedades não guardam diretamente o valor dos atributos do objetos, apenas o nome do atributo para fazer a recuperação do valor posteriormente com `Reflections` usando `BeanUtils`.

A anotação:

```

* @ValorZeroIncluiDescricao.java x Restaurante.java x ValorZeroIncluiDescricaoValidator.java x
7   import java.lang.annotation.RetentionPolicy;
8   import java.lang.annotation.Target;
9
10  4 usages
11  @Target({ElementType.TYPE})/*Somente em classes, interfaces, enum, records*/
12  @Retention(RetentionPolicy.RUNTIME)
13  @Constraint(validatedBy = {ValorZeroIncluiDescricaoValidator.class})
14  public @interface ValorZeroIncluiDescricao {
15      String message() default "descrição obrigatória inválida." +
16          " Deve conter 'Frete Grátis' no nome do restaurante";
17
18      Class<?>[] groups() default {};
19
20      Class<? extends Payload>[] payload() default {};
21
22      2 usages
23      String valorField();
24      2 usages
25      String descricaoField();
26      2 usages
27      String descricaoObrigatoria();
28  }

```

A declaração de uma anotação customizada do Bean Validation é padrão e apenas mudamos os atributos da anotação, mensagem de erro e a classe constraint.

Classe de validação (constraint class):

```

1 usage
public class ValorZeroIncluiDescricaoValidator implements
    ConstraintValidator<ValorZeroIncluiDescricao, Object> {

    2 usages
    private String valorField;
    3 usages
    private String descricaoField;
    3 usages
    private String descricaoObrigatoria;

    @Override
    public void initialize(ValorZeroIncluiDescricao constraintAnnotation) {...}

    /**
     *
     * @param objetoValidacao object to validate - objeto da classe na qual está anotada a validacao
     * @param context context in which the constraint is evaluated
     */
    @Override
    public boolean isValid(Object objetoValidacao, ConstraintValidatorContext context) {...}
}

```

a classe da constraint tem que implementar ConstraintValidator e passar nos argumentos do tipo a própria anotação e o tipo do objeto a ser validado. Como é uma anotação a nível de classe e anotamos na declaração da classe, vamos adicionar Object. se fosse apenas validação de String, seria uma String, de números poderia ser Number. Além disso, é

necessário sobrescrever dois métodos Initialize e isValid.

Método initialize: tem como parâmetro a anotação e serve para capturar os valores das propriedades da anotação na hora da declaração da classe.

Método isValid: 1º argumento é o tipo da classe, e o segundo é um ConstraintValidatorContext.

método Initialize:

```
public class ValorZeroIncluiDescricaoValidator implements
    ConstraintValidator<ValorZeroIncluiDescricao, Object> {

    2 usages
    private String valorField;
    3 usages
    private String descricaoField;
    3 usages
    private String descricaoObrigatoria;

    @Override
    public void initialize(ValorZeroIncluiDescricao constraintAnnotation) {
        this.valorField = constraintAnnotation.valorField();
        this.descricaoField = constraintAnnotation.descricaoField();
        this.descricaoObrigatoria = constraintAnnotation.descricaoObrigatoria();
        ConstraintValidator.super.initialize(constraintAnnotation);
    }
}
```

podemos adicionar valores aos atributos do validator através da anotação recebida no parâmetro.

método isValid:

```

30      @Override
31      public boolean isValid(Object objetoValidacao, ConstraintValidatorContext context) {
32          boolean isValid = true;
33          System.out.println(descricaoObrigatoria);
34          System.out.println(descricaoField);
35
36          try {
37              final BigDecimal taxaFrete = (BigDecimal) BeanUtils
38                  .getPropertyDescriptor(objetoValidacao.getClass(), valorField)
39                  .getReadMethod().invoke(objetoValidacao);
40
41              final String nome = (String) BeanUtils
42                  .getPropertyDescriptor(objetoValidacao.getClass(), descricaoField)
43                  .getReadMethod().invoke(objetoValidacao);
44
45              if(taxaFrete != null && BigDecimal.ZERO.compareTo(taxaFrete) == 0
46                  && nome != null){
47                  isValid = nome.toLowerCase().contains(descricaoObrigatoria.toLowerCase());
48              }
49
50          } catch (Exception e) {
51              throw new ValidationException(e);
52          }
53
54          return isValid;
55      }
56  }
57

```

A lógica é a seguinte: capturar o valor do atributo taxaFrete daquele objeto Restaurante usando Reflections com BeanUtils, capturar o valor do atributo nome do mesmo objeto e fazer uma validação "na mão" se taxaFrete é diferente de null e a constante ZERO(0) é igual ao valor de taxaFrete(retorna 0 se os valores forem iguais) e se nome é diferente de nulo, se toda a condição for true, o nome do restaurante e o nome obrigatório são convertidos para minúsculo e verificado se em nome contém em alguma parte a descrição obrigatória.

BeanUtils:

a ideia é chamar o método get do objeto para capturar os valores necessário, então chamamos BeanUtils e usando o método getPropertyDescriptor, passando 2 argumentos, 1º a classe do objeto e o 2º, o nome do atributo para executar as ações.

valorField=taxaFrete

objetoValidacao é do tipo Object definido no parâmetro do método, mas sabemos que é do tipo Restaurante, pois anotamos na classe, porém, poderia ser de outros tipos e teríamos que fazer uma condição com instanceof e um cast implícito.

The screenshot shows a REST client interface. At the top, a POST request is configured to `http://localhost:8080/restaurantes`. The request body is in JSON format, containing the following data:

```
{
  "nome": "Colombiana Frete",
  "taxaFrete": 0.00,
  "cozinha": {
    "id": 1
  }
}
```

The response is a 400 Bad Request, with a status of 400, a type of `http://localhost:8080/dados-invalidos`, a title of "Dados inválidos", and a detail message: "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente." The response also includes a `userMessage` field with the same detail message, a `timestamp` of "2023-03-05T22:44:51.8383392", and an empty `fields` array.

```
{
  "status": 400,
  "type": "http://localhost:8080/dados-invalidos",
  "title": "Dados inválidos",
  "detail": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
  "userMessage": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
  "timestamp": "2023-03-05T22:44:51.8383392",
  "fields": []
}
```

não foi validado pois o frete tem valor 0.00 e o nome não contém "Frete Grátis". O array de fields está vazio, não recebemos a mensagem específica do problema e seu campo. Vamos corrigir isso em [9.18. Ajustando Exception Handler para adicionar mensagens de validação em nível de classe](#)

9.18. Ajustando Exception Handler para adicionar mensagens de validação em nível de classe

domingo, 5 de março de 2023 22:48

Tecnicamente, temos duas propriedades que geram erros, mas no corpo da resposta da exceção, a lista de fields retorna vazia.

The screenshot shows a REST client interface. At the top, a POST request is configured to `http://localhost:8080/restaurantes`. The request body is in JSON format and contains the following data:

```
{
  "nome": "Colombiana Frete",
  "taxaFrete": 0.00,
  "cozinha": {
    "id": 1
  }
}
```

The response is a 400 Bad Request with a status of 400, a response time of 16 ms, and a size of 471 B. The response body is shown in JSON format and contains the following data:

```
{
  "status": 400,
  "type": "http://localhost:8080/dados-invalidos",
  "title": "Dados inválidos",
  "detail": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
  "userMessage": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
  "timeStamp": "2023-03-05T22:44:51.8383392",
  "fields": []
}
```

na classe `ApiHandlerException` que contém o método que trata exceções de sintaxe, fazemos um `.stream.map` para iterar a lista de `FieldErros`


```

1 usage
@Override
protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex, HttpHeaders headers,
    HttpStatus status, WebRequest request) {

    final List<Problem.Field> problemFieldErrors = ex.getFieldErrors().stream().map(fd -> {
        String message = messageSource.getMessage(fd, LocaleContextHolder.getLocale());
        return Problem.Field.builder()
            .name(fd.getField())
            .userMessage(message)
            .build();
    }).toList();

    final ProblemType problemType = ProblemType.DADOS_INVALIDOS;
    String detail = "Um ou mais campos estão invalidados. Faça o preenchimento e" +
        " tente novamente.";
    final Problem problema = createProblemType(status, problemType, detail)
        .userMessage(detail)
        .fields(problemFieldErrors)
        .build();
    return handleExceptionInternal(ex, problema, headers, status, request);
}

```

Quando colocamos a anotação na classe, não é um erro de Field, mas de Object, porque não estamos tratando individualmente atributos da entidade.

podemos refatorar o trecho do método para ex.getAllErrors e no map, para cada objeto da lista, verificar se é um ObjectError ou FieldError

```

@Override
protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex, HttpHeaders headers,
    HttpStatus status, WebRequest request) {

    final List<Problem.Objects> problemObjectsErrors = ex.getAllErrors().stream().map(fd -> {
        String message = messageSource.getMessage(fd, LocaleContextHolder.getLocale());
        String name = null; //fd.getField()

        if(fd instanceof FieldError fieldError){
            name = (fieldError).getField();
        }
        name = fd.getObject().getName();
        return Problem.Objects.builder()
            .name(name)
            .userMessage(message)
            .build();
    }).toList();
}

```

```

// Se uma anotação a nível de classe é preenchida, o erro é de tipo
ValorZeroIncluiDescricao=0 campo {1} deve conter a palavra {2}.

```


POST

http://localhost:8080/restaurantes

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

Beautify

```
1 {
2   ... "nome": "Colombiana Frete",
3   ... "taxaFrete": 0.00,
4   ... "cozinha": {
5     ... "id": 1
6   }
7 }
```

Body

Cookies

Headers (4)

Test Results

Status: 400 Bad Request

Time: 31 ms

Size: 562 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

7

8

9

10

11

12

13

14

```
{
  "status": 400,
  "type": "http://localhost:8080/dados-invalidos",
  "title": "Dados inválidos",
  "detail": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
  "userMessage": "Um ou mais campos estão invalidados. Faça o preenchimento e tente novamente.",
  "timeStamp": "2023-03-06T11:14:00.4695136",
  "objects": [
    {
      "name": "restaurante",
      "userMessage": "O campo nome deve conter a palavra Frete Grátis. "
    }
  ]
}
```

9.19. Executando processo de validação programaticamente (PATCH)

segunda-feira, 6 de março de 2023 00:12

Ao atualizar via PATCH, recebemos no método um Map de propriedade:valor do corpo da requisição contendo as propriedades para a atualização do Restaurante. Normalmente o Spring faz o binding de valores vindo da requisição para o objeto na assinatura do método, como passar as propriedades de restaurante no corpo e receber um objeto restaurante.

```
@PostMapping
public ResponseEntity<?> adicionar
    (@RequestBody @Valid Restaurante restaurante) {
```

e o Spring consegue fazer a validação com Bean Validation. Mas não é o caso quando recebemos as propriedades no formato de uma coleção. Temos que fazer a validação programaticamente.

```
@PatchMapping("/{id}")
public Restaurante atualizarParcial(@PathVariable Long id, @RequestBody Map<String, Object> campos,
    HttpServletRequest servletRequest) {
    Restaurante restauranteAtual = restauranteService.buscar(id);
```

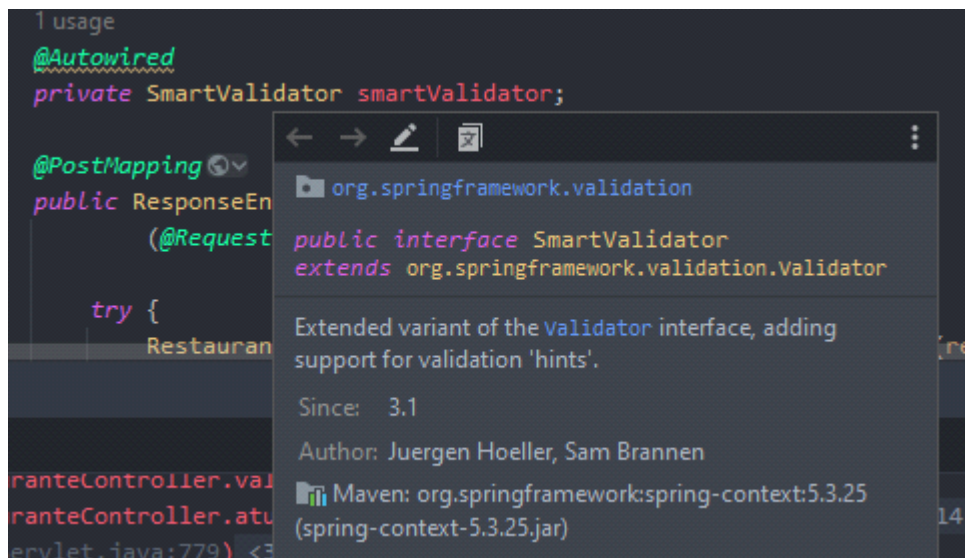
Logo após tratar os dados podemos chamar o método `validate` passando o objeto da validação e o nome da entidade, no caso restaurante, e poderemos aplicar as regras de validação:

```
@PatchMapping("/{id}")
public Restaurante atualizarParcial(@PathVariable Long id, @RequestBody Map<String, Object> campos,
    HttpServletRequest servletRequest) {
    Restaurante restauranteAtual = restauranteService.buscar(id);

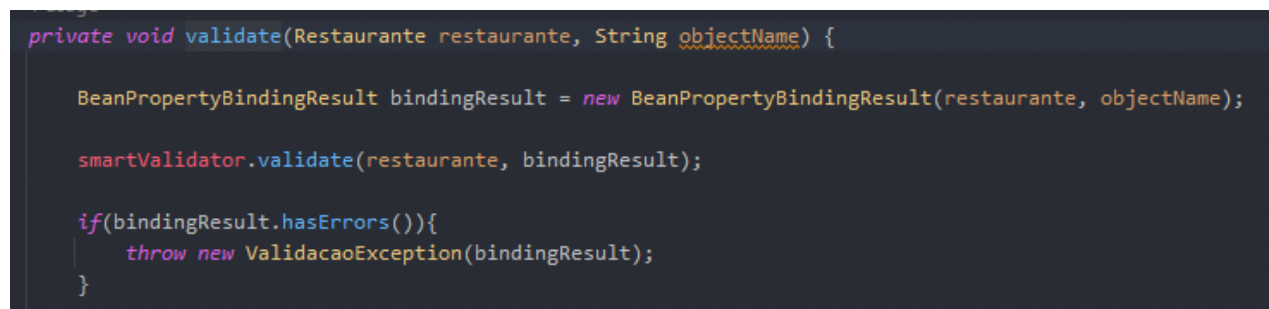
    merge(campos, restauranteAtual, servletRequest);
    validate(restauranteAtual, objectName: "restaurante");

    return atualizar(restauranteAtual, id);
}
```

Primeiro temos que injetar uma instância de SmartValidator

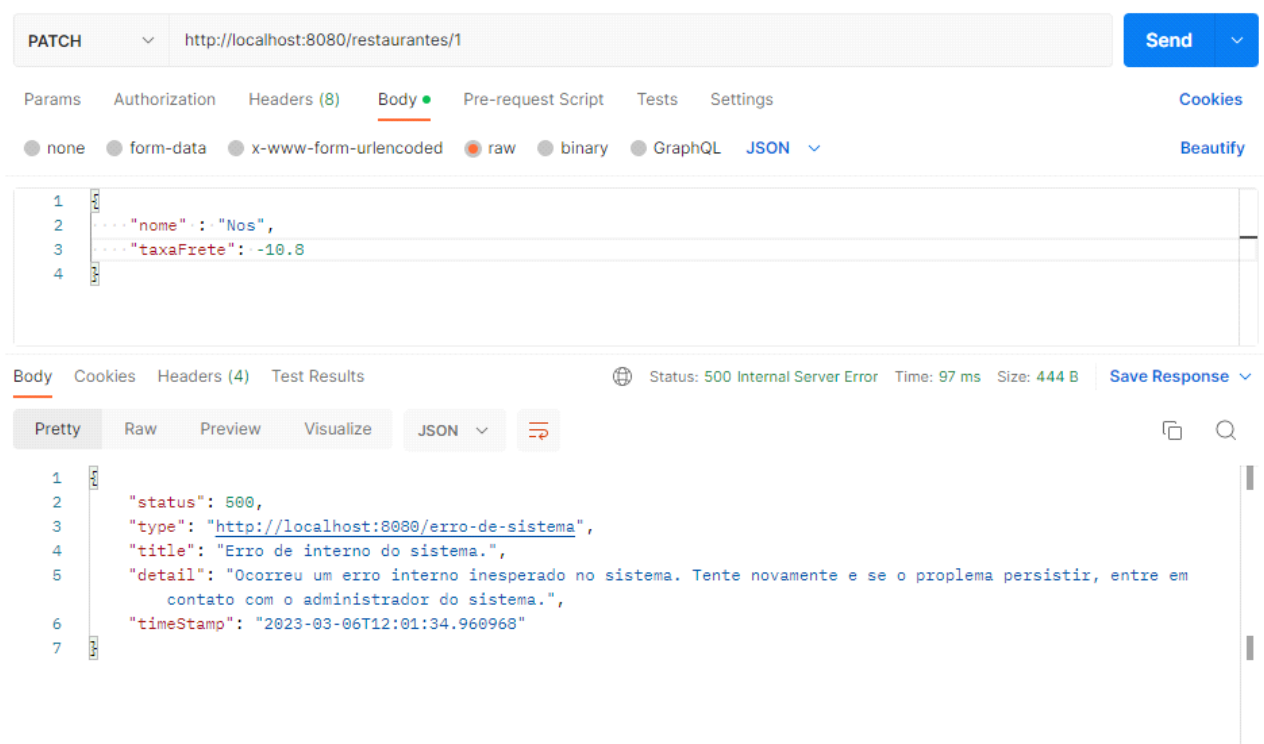


para que possamos chamar o método `.validate` do `SmartValidator`, o 1º argumento recebe um objeto que queremos validar e o 2º espera um `Error`, podemos ter uma instância desse tipo através de um objeto tipo `BeanPropertyBindingResult` que é um `Error`.



caso há erros, chamamos a nossa exceção

que recebe o `bindingResult`



Temos um erro no atributo `taxaFrete` que possui a validação `@PositiveOrZero`, mas ainda temos que tratar os campos de erro.

9.20. Desafio tratando a exception customizada de validações programáticas

segunda-feira, 6 de março de 2023 13:57