AULA 15 - VALIDAÇÃO DE DADOS

Disciplina de Backend - Professor Ramon Venson - SATC 2024

Validação de Dados

A validação de dados é o processo de verificação dos dados que atendem aos requisitos, comparando-os com um conjunto de regras que já foram configuradas ou definidas.

Importância da Validação

- Qualidade de dados (acurácia e confiabilidade)
- Prevenção de erros
- Consistência nos dados entre registros
- Compliance (razões legais)
- Velocidade de processamento
- Abstração de dados

Objetivos da Validação

Validar:

- campos obrigatórios;
- campos não permitidos;
- tipo de dados;
- range dos dados;
- formato dos dados;
- unicidade dos dados;

Contras da Validação

- Complexidade;
- Manutenção de código;
- Erros de validação;
- Mudança de necessidades;

Jakarta Validation API

- Especificação JSR 380
- Provê um conjunto de anotações para validação automática de beans
- Excessões disparadas automaticamente quando um conjunto não é validado

Uma validação pode ser realizada adicionando uma `@Anotação` sobre qualquer um dos atributos de um bean/classe.

Hibernate Validation

O Hibernate Validation é a implementação de referência do JSR 380 e implementa a validação de dados.

Podemos incluí-la no projeto usando o Spring Initializr ou adicionando essa dependência no arquivo maven:

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

Exemplo de validação

```
public class Usuario {
    @NotNull(message = "O nome não pode ser nulo")
    private String name;

    @Size(min = 10, max = 200, message
      = "Descrição precisa ser entre 10 e 20 caracteres")
    private String aboutMe;

    @Min(value = 18, message = "Idade precisa ser maior que 18 anos")
    @Max(value = 150, message = "Idade precisa ser menor que 150 anos")
    private int age;

    @Email(message = "Email precisa ser válido")
    private String email;
}
```

A maioria das anotações podem receber como parâmetro um atributo `message`, que corresponde à mensagem a ser retornada quando o atributo não é validado corretamente.

Outros atributos podem ser específicos para cada anotação, como o caso do `value` disponível para as anotações `@Min` e `@Max`.

Todas as anotações

Validando DTOs

Para validar DTOs, usamos a mesma estrutura utilizada em qualquer tipo de bean.

Podemos usar o controller que recebe um DTO para validar automaticamente os atributos do corpo da requisicao. Para isso, usamos a anotação `@Valid`.

```
@RestController
public class TransacaoController {
    @PostMapping("/noticia")
    public String postNoticia(@RequestBody @Valid NoticiaDto noticiaDto) {
      // Caso aconteça algum erro ne validação,
      // o controlador dispara uma exception
    }
```

Anotações frequentes

Anotação	Descrição
`@AssertFalse`	Atributo precisa ser falso (`Boolean` ou `bool`)
`@AssertTrue`	Atributo precisa ser verdadeiro (`Boolean` ou `bool`)
`@Digits`	Número de dígitos do número (especificado com `integer` e `fraction`)
`@Email`	Atributo precisa ser email valido
`@Future`	Atributo precisa ser uma data no futuro
`@FutureOrPresent`	Atributo precisa ser uma data atual ou no futuro

Anotação	Descrição
`@Max`	Atributo precisa ser número máximo (especificado com `value`)
`@Max`	Atributo precisa ser número mínimo (especificado com `value`)
`@Negative`	Atributo precisa ser número negativo
`@NegativeOrZero`	Atributo precisa ser número zero ou negativo
`@NotBlank`	Atributo string precisa conter pelo menos um caracter que não seja espaço
`@NotEmpty`	Lista não pode ser vazia

Anotação	Descrição
`@NotNull`	Atributo não pode ser vazio
`@Null`	Atributo precisa ser nulo
`@Past`	Atributo precisa ser uma data passada
`@PastOrPresent`	Atributo precisa ser uma data atual ou passada
`@Positive`	Atributo precisa ser número positivo
`@PositiveOrZero`	Atributo precisa ser número zero ou positivo
`@Size`	O elemento precisa ter o tamanho especificado (vetores ou strings - especificado com parâmetros `min` e `max`)

Validando com Expressões Regulares

Por fim, também temos a anotação `@Pattern` que permite definir uma expressão regular, geralmente utilizada para especificar um padrão de texto.

Aqui temos um exemplo do uso do `@Pattern` para especificar um CEP:

```
@Pattern(regexp="\\d{5}-\\d{3})
private String cep;
```

O termo `regexp` significa ***Reg**ular **Ex**pression **P**attern*. Os termos `\d{5}` e `\d{3}` significam, respectivamente, "Cinco dígitos" e "Três digitos". Isso faz com que qualquer String que contenha não contenha esse padrão seja rejeitada.

Mais informações sobre expressões regulares no Aurélio.net.

Repassando exceções para o cliente

As mensagens de exceção retornadas são geralmente exibidas no console. Para repassar as mensagens de erro para o cliente da API, devemos manipular a exceção criando um controlador global específico para esse propósito:

```
@RestControllerAdvice
public class GlobalExceptionHandler {
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public List<ApiExceptionDto> handleValidationExceptions(MethodArgumentNotValidException ex) {
        return ex.getBindingResult().getAllErrors().stream().map((error) -> {
            return new ApiExceptionDto("Erro de validação", error.getDefaultMessage(), Instant.now());
        }).toList();
    }
}
```

O que aprendemos hoje

- Identificar a necessidade de validação de dados;
- Usar a especificação Jakarta Validation API para validar um bean;
- Usar expressões regulares para validar padrões em strings;
- Criar um controlador de exceções globais para as validações.