

FIAP

Análise e Desenvolvimento de Sistemas

Prof. MSc Marcelo Tomio Hama

AULA 7

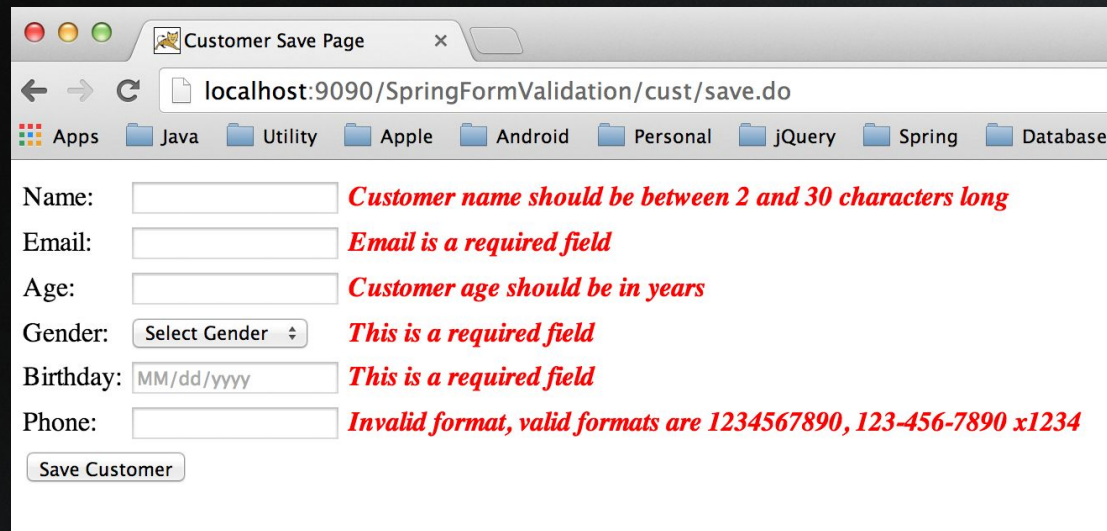
API Restful com Spring Data JPA e Spring Validation.

OBJETIVOS

1. Checar e validar exercícios práticos realizados em aulas anteriores;
2. Entendimento do Spring Validation e suas anotações;
3. Exercícios práticos com o Spring Validation em forms.

Spring Validation

Validar a entrada do usuário é um requisito muito comum na maioria dos aplicativos, e a estrutura Java Bean Validation se tornou o padrão de fato para lidar com esse tipo de lógica.



The screenshot shows a web browser window titled 'Customer Save Page' at the URL 'localhost:9090/SpringFormValidation/cust/save.do'. The browser's file explorer shows a path: Apps > Java > Utility > Apple > Android > Personal > jQuery > Spring > Database. The form contains the following fields and messages:

- Name: *Customer name should be between 2 and 30 characters long*
- Email: *Email is a required field*
- Age: *Customer age should be in years*
- Gender: *This is a required field*
- Birthday: *This is a required field*
- Phone: *Invalid format, valid formats are 1234567890, 123-456-7890 x1234*

A 'Save Customer' button is located at the bottom left of the form.

JSR 380

Uma especificação da API Java para validações, parte do Jakarta EE e JavaSE.

Garante que as propriedades de um bean atendam a critérios específicos através de anotações. Requer Java 17 ou superior (Spring Boot 3.x + Hibernate-Validator 8.0.0).

```
<dependency>
  <groupId>org.hibernate.validator</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>8.0.0.Final</version>
</dependency>
```


Anotações do Spring Validation

@NotNull verifica que o valor da propriedade anotada não é nulo.

@AssertTrue verifica que o valor da propriedade anotada é verdadeiro.

@Size verifica que o valor da propriedade anotada tem um tamanho entre os atributos “min” e “max”. Podemos aplicá-lo às propriedades String, Collection, Map e Array.

@Min verifica que a propriedade anotada tem um valor não menor que o atributo value.

@Max verifica que a propriedade anotada tem um valor não maior que o atributo value.

@Email verifica que a propriedade anotada é um endereço de email válido.

@NotEmpty verifica que a propriedade não é nula ou vazia. Podemos aplicá-lo a valores String, Collection, Map ou Array.

@NotBlank apenas para textos, verifica que a propriedade não é nula ou espaço em branco.

@Positive e **@PositiveOrZero** apenas valores numéricos, verifica se são estritamente positivos ou positivos incluindo 0.

@Negative e **@NegativeOrZero** apenas valores numéricos, verifica se são estritamente negativos ou negativos incluindo 0.

@Past e **@PastOrPresent** verifica que um valor de data está no passado ou no passado incluindo o presente.

@Future e **@FutureOrPresent** validam que um valor de data está no futuro ou no futuro incluindo o presente.

Exemplos com Spring Validation

```
public class User {  
  
    @NotNull(message = "Name cannot be null")  
    private String name;  
  
    @AssertTrue(message = "Working must be true")  
    private boolean working;  
  
    @Size(min = 10, max = 200, message  
        = "About Me must be between 10 and 200 characters")  
    private String aboutMe;  
  
    @Min(value = 18, message = "Age should not be less than 18")  
    @Max(value = 150, message = "Age should not be greater than 150")  
    private int age;  
  
    @Email(message = "Email should be valid")  
    private String email;  
  
    // standard setters and getters  
}
```

```
private LocalDate dateOfBirth;  
  
public Optional<@Past LocalDate> getDateOfBirth() {  
    return Optional.of(dateOfBirth);  
}
```

Hands-On Laboratorial

Formulário de Cadastro com Validação

Para este exercício, vamos criar um form que recebe valores inseridos para realizar cadastro de clientes. As novas tecnologias que iremos ter contato são:

- **Thymeleaf**: um template engine para facilitar a construção de páginas HTML;
- **Form HTML**: iremos construir uma primeira página, simples, para receber dados;
- **Controller**: um controller MVC específico para aplicações web;
- **Spring Validation**: mecanismos de validação de campos do Spring Framework.

Passo-a-Passo

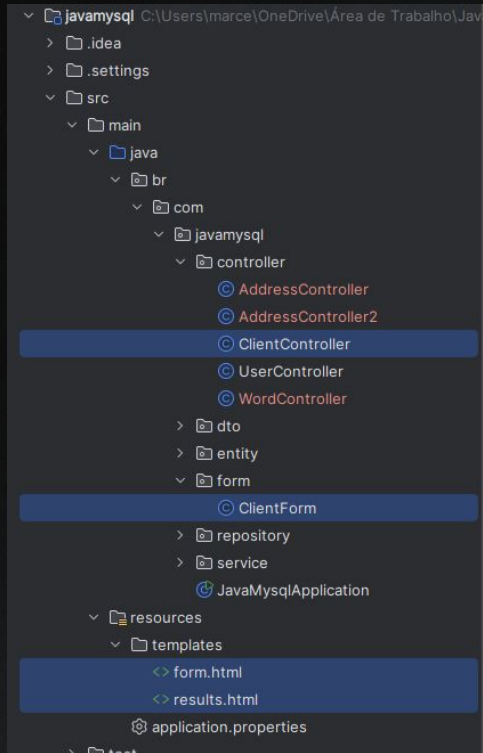
1. Implemente a classe modelo que representa a view a ser validada;
2. Implemente os templates html para exibição do frontend;
3. Implemente o controller que servirá para direcionar e controlar os métodos HTTP.

Curiosidade: abra e olhe o arquivo `ValidationMessages_pt_BR.properties`.



Hands-On Laboratorial

Estrutura do Projeto e Imports



```
<dependencies>

    <dependency>
        <groupId>org.hibernate.validator</groupId>
        <artifactId>hibernate-validator</artifactId>
        <version>8.0.0.Final</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>

</dependencies>
```

Hands-On Laboratorial

Classe modelo para representar a view

```
public class ClientForm {  
  
    @NotNull  
    @Size(min=2, max=30)  
    private String name;  
  
    @NotNull  
    @Min(18)  
    private Integer age;  
  
    public String toString() {  
        return "Client(Name: " + this.name + ", Age: " + this.age + ")";  
    }  
  
}
```



Hands-On Laboratorial

Templates HTML

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<body>
<form action="#" th:action="@{/}" th:object="${clientForm}" method="post">
  <table>
    <tr>
      <td>Name:</td>
      <td><input type="text" th:field="*{name}" /></td>
      <td th:if="${#fields.hasErrors('name')}" th:errors="*{name}">Name Error</td>
    </tr>
    <tr>
      <td>Age:</td>
      <td><input type="text" th:field="*{age}" /></td>
      <td th:if="${#fields.hasErrors('age')}" th:errors="*{age}">Age Error</td>
    </tr>
    <tr>
      <td><button type="submit">Submit</button></td>
    </tr>
  </table>
</form>
</body>
</html>
```

```
<html>
<body>
  Congratulations! You are old enough to sign up for this site.
</body>
</html>
```

Hands-On Laboratorial

Controller HTTP

```
@Controller| no usages
public class ClientController implements WebMvcConfigurer {

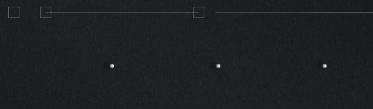
    @Override 2 usages
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController(urlPathOrPattern: "/results").setViewName("results");
    }

    @GetMapping("/") no usages
    public String showForm(ClientForm clientForm) {
        return "form";
    }

    @PostMapping("/") no usages
    public String checkClientInfo(@Valid ClientForm clientForm, BindingResult bindingResult) {

        if (bindingResult.hasErrors()) {
            return "form";
        }

        return "redirect:/results";
    }
}
```



Hands-On Laboratorial



← → ↻ ⓘ localhost:8080

Name:

Age: deve ser maior que ou igual à 18

Crie campos adicionais para serem inseridos e validados

1. Nome (não nulo, menor que 64 caracteres)
2. Sobrenome (não nulo, menor que 64 caracteres)
3. Email (endereço de email válido)
4. Endereço, com rua, número, bairro, cidade, estado, e CEP, cada qual com seu validador

Tema para casa

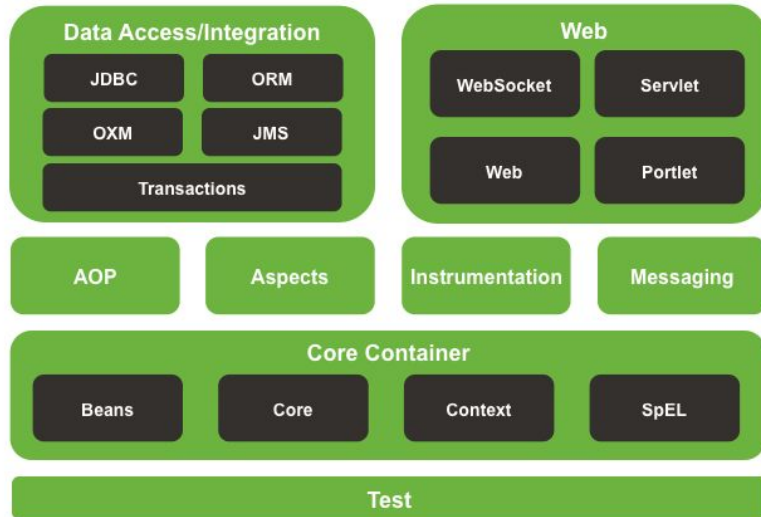
Crie o campo gênero com um validation enumerável.

Exemplo: <https://www.baeldung.com/javax-validations-enums>

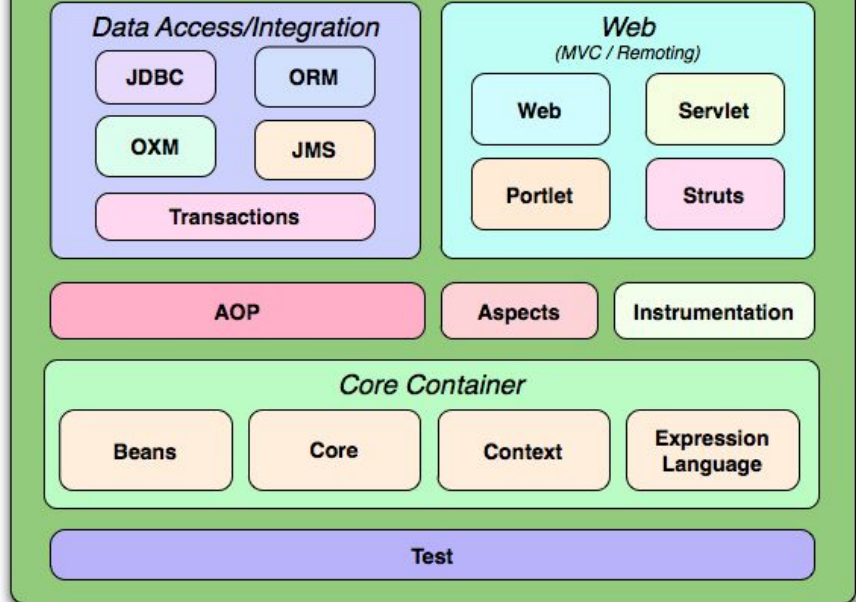
Framework Spring



Spring Framework Runtime



Spring Framework Runtime



Execução do Spring Framework. Fonte:

<https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/spring-introduction.html>

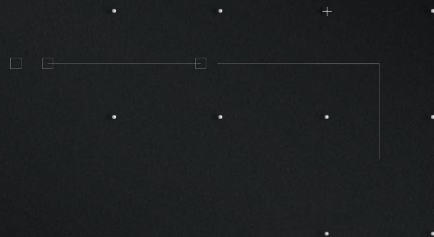
Referências

Email do Professor

profmarcelo.hama@fiap.com.br

Bibliografias/Sites

- <https://www.baeldung.com/java-validation>
<https://spring.io/guides/gs/validating-form-input>





*"A educação tem raízes
amargas, mas os seus frutos
são doces."*

Aristóteles

FIM
