Bean Validation

DSC AULA 6

Bean Validation

- A API de Bean Validation fornece uma facilidade para validar objetos em diferentes camadas da aplicação.
  - Com Bean Validation, as restrições ficam concentradas nas classes de negócio, em forma de anotações.
  - Bean Validation é uma especificação. As implementações podem adicionar restrições customizadas, além das fornecidas pela especificação.
  - Usaremos o Hibernate Validator como implementação.

Bean Validation

- Observe as dependências ligadas ao Hibernate Validator, que vamos utilizar

```
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>8.0.0.Final</version>
</dependency>
<dependency>
    <groupId>org.glassfish.expressly</groupId>
    <artifactId>expressly</artifactId>
    <version>5.0.0</version>
</dependency>
```

CartaoCredito

- Observe as anotações @NotNull, @NotBlank, @Size, @Future
  - Elas são restrições da API Bean Validation
- Observe a anotação @CreditCardNumber
  - Ela é uma restrição específica do Hibernate Validator

```
public class CartaoCredito implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @NotNull
    @OneToOne(mappedBy = "cartaoCredito", optional = false)
    private Comprador dono;
    @NotBlank
    @Size(max = 15)
    @Column(name = "TXT_BANDEIRA")
    private String bandeira;
    @NotNull
    @CreditCardNumber
    @Column(name = "TXT_NUMERO")
    private String numero;
    @NotNull
    @Future
    @Temporal(TemporalType.DATE)
    @Column(name = "DT_EXPIRACAO", nullable = false)
    private Date dataExpiracao;
```

Bean Validation

@NotNull	O valor não pode ser null.
@NotBlank	O string não pode ser null nem vazio, strings como “ ” não são válidos.
@Size	O tamanho do valor da propriedade deve estar entre os limites configurados. Funciona com string, coleção, mapa e array.
@CreditCardNumber	Utiliza o algoritmo Luhn (http://en.wikipedia.org/wiki/Luhn_algorithm) para validar um nº de cartão de crédito. Essa restrição é específica do Hibernate Validator.
@Future	A data precisa ser futura.

Usuario

- Observe restrição @CPF, que é específica do Hibernate Validator, bem como as restrições @Email e @Past, que são padrão Bean Validation.
    - A restrição @Past é aplicada a uma data, que, para ser válida, precisa ser uma data passada.

```
@NotNull
@CPF
@Column(name = "TXT_CPF")
protected String cpf;

@NotNull
@Email
@Column(name = "TXT_EMAIL")
protected String email;

@Past
@Temporal(TemporalType.DATE)
@Column(name = "DT_NASCIMENTO")
protected Date dataNascimento;
```

Usuario

- Observe a restrição @Pattern, que permite a construção de expressões regulares
  - Neste caso, o *primeiroNome* deve ter o seguinte padrão: uma letra maiúscula, seguida de pelo menos uma letra minúscula.
  - Para mais detalhes sobre padrões, consulte https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html

```
@NotBlank
@Size(max = 30)
@Pattern(regexp = "\\p{Upper}{1}\\p{Lower}+", message = "{exemplo.jpa.Usuario.nome}")
@Column(name = "TXT_PRIMEIRO_NOME")
protected String primeiroNome;
```

Usuario

- Observe, ainda, a expressão regular da senha, que deve possuir, pelo menos, em qualquer ordem: um dígito, uma letra minúscula, uma letra maiúscula e um sinal de pontuação.
    - Além disso, a senha deve possuir entre 6 e 20 caracteres.

```
@NotBlank
@Size(min = 6, max = 20)
@Pattern(regexp = "((?=.*\\p{Digit})(?=.*\\p{Lower})(?=.*\\p{Upper})(?=.*\\p{Punct}).{6,20})",
        message = "{exemplo.jpa.Usuario.senha}")
@Column(name = "TXT_SENHA")
protected String senha;
```

Bean Validation

@CPF	Valida um CPF. Essa restrição é específica do Hibernate Validator.
@Email	Valida um e-mail.
@Past	A data precisa ser do passado.
@Pattern	A data precisa ser futura.

Existem outras restrições. Para conhecê-las, consulte https://hibernate.org/validator/.

ValidationMessages_pt_BR.properties

- Observe que, na anotação @Pattern, há uma mensagem
  - message = "{exemplo.jpa.Usuario.nome}"
  - Essa mensagem deve ser escrita em uma arquivo de mensagens chamado ValidationMessages_pt_BR.properties
    - Esse arquivo deve ser posto na raiz do Classpath, como pode ser visto no exemplo_08

```
@NotBlank
@Size(max = 30)
@Pattern(regexp = "\\p{Upper}{1}\\p{Lower}+", message = "{exemplo.jpa.Usuario.nome}")
@Column(name = "TXT_PRIMEIRO_NOME")
protected String primeiroNome;
```

ValidationMessages_pt_BR.properties

- O arquivo ValidationMessages_pt_BR.properties contendo mensagens customizadas.

```
exemplo.jpa.Usuario.nome = Deve possuir uma única letra maiúscula, seguida por letras minúsculas.
exemplo.jpa.Usuario.senha = A senha deve possuir pelo menos um caractere de: pontuação, maiúscula, minúscula e número.
exemplo.jpa.Endereco.cep = CEP inválido. Deve estar no formado NN.NNN-NNN, onde N é número natural.
exemplo.jpa.Endereco.estado = Estado inválido.
```

Criando Uma Nova Restrição

- Observe abaixo a validação customizada @ValidaEstado

```
@NotBlank
@ValidaEstado
@Size(min = 2, max = 2)
@Column(name = "END_TXT_ESTADO")
private String estado;
```

Criando Uma Nova Restrição

- Observe que devemos criar a anotação.
  - Ela é bastante padrão.
  - Importante definir uma mensagem padrão, que deve ser posta do arquivo de propriedades (no nosso caso *ValidationMessages_pt_BR.properties*)

```
@Target( {ElementType.FIELD, ElementType.METHOD, ElementType.ANNOTATION_TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = ValidadorEstado.class)
@Documented
public @interface ValidaEstado {
    String message() default "{exemplo.jpa.Endereco.estado}";

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

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

Criando Uma Nova Restrição

- A lógica da validação deve ficar numa classe que implementa *ConstraintValidator*

```
public class ValidadorEstado implements ConstraintValidator<ValidaEstado, String> {
    private List<String> estados;

    @Override
    public void initialize(ValidaEstado validaEstado) {...30 lines }

    @Override
    public boolean isValid(String valor, ConstraintValidatorContext context) {
        return valor == null ? false : estados.contains(valor);
    }

}
```

Para Baixar

- O código completo do exemplo 8, acesse
  - https://svn.riouxsvn.com/dsc-ifpe/exemplo_08/