Desenvolvimento WEB

-

Full Stack Completo: Java + React

Prof. Roberto Alencar

Validações no Back-end

Validações no Back-end

- Erros ocorrem e sempre irão ocorrer, seja por uma especificação errada nos requisitos do sistema, seja por erros de desenvolvimento ou qualquer outra razão. O fato é que erros acontecem e precisamos saber lidar com eles.

- Mas como vamos tratar e apresentar o erro para o usuário ou cliente que chamam as nossas APIs?

- Nesse contexto, iremos estudar nesta aula 3 tipos de validações:

    - Validações na Entrada das Requisições;
    - Validações no Banco de Dados;
    - Validações de Regras de Negócio.

Validações
na Entrada das Requisições

Validações no Back-end :: Entrada das Requisições

- Vamos entender a princípio como seria a implementação de validações sem o uso de uma biblioteca, ou seja, implementando as validações “a moda antiga”.
- Por exemplo, caso precisássemos implementar as seguintes validações no nosso cadastro de clientes:
  - O campo `nome` não pode ser nulo e nem vazio (“”);
  - O campo `nome` não pode ser maior que 100 caracteres;
  - O campo `CPF` não pode ser nulo e nem vazio (“”);
  - O campo `CPF` precisar ser informado com um cpf válido;
  - O campo `foneCelular` tem que ter mais que 8 caracteres e menos que 20;

Validações no Back-end :: Entrada das Requisições

Como estava nosso método `save` **antes** das validações na classe `ClienteController`:

```
@PostMapping
public ResponseEntity<Cliente> save(@RequestBody @Valid ClienteRequest request) {

     Cliente cliente = clienteService.save(request.build());
     return new ResponseEntity<Cliente>(cliente, HttpStatus.CREATED);
}
```

Validações no Back-end :: Entrada das Requisições

Como estava nosso método `save` **antes** das validações na classe `ClienteController`:

```
@PostMapping
public ResponseEntity<Cliente> save(@RequestBody @Valid ClienteRequest request) {

        Cliente clienteRequisicao = request.build();
        StringBuilder erros = new StringBuilder();

        //O campo nome não pode ser nulo e nem vazio
        if (clienteRequisicao.getNome() == null || clienteRequisicao.getNome().equals("")) {
            erros.append("O campo Nome é de preenchimento obrigatório. ");
        }

        //O campo nome não pode ser maior que 100 caracteres
        if (clienteRequisicao.getNome() != null && clienteRequisicao.getNome().length() > 100) {
            erros.append("O campo Nome não pode ter mais que 100 caracteres. ");
        }

        //O campo CPF não pode ser nulo e nem vazio
        if (clienteRequisicao.getCpf() == null || clienteRequisicao.getCpf().equals("")) {
            erros.append("O campo CPF é de preenchimento obrigatório. ");
        }

        //O campo CPF precisar ser informado com um cpf válido;
        // ...

        //O campo fone tem que ter mais que 8 caracteres e menos que 20
        if (clienteRequisicao.getFone() != null && (clienteRequisicao.getFone().length() < 8 || clienteRequisicao.getFone().length() > 20)) {
            erros.append("O campo Fone tem que ter entre 8 e 20 caracteres. ");
        }

        if (erros.length() > 0) {
            throw new BadRequestException(erros.toString());
        }

        Cliente clienteSalvo = clienteService.save(clienteRequisicao);
        return new ResponseEntity<Cliente>(clienteSalvo, HttpStatus.CREATED);
}
```

Validações no Back-end :: Entrada das Requisições

É bem trabalhoso não é ?

Validações no Back-end :: Entrada das Requisições

- Para resolver esse problema, o Spring fornece um mecanismo para a tratativa de erros que é muito válido e facilita, e muito, o desenvolvimento de APIs, ele se chama **Spring Validation**.

- Essa biblioteca é adicionada* ao projeto através da inclusão da dependência abaixo no arquivo `pom.xml`:

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

* Atenção: essa biblioteca já foi adicionado no momento da criação do projeto

Validações no Back-end :: Entrada das Requisições

Desta forma, todo o código em vermelho do Controller pode ser substituído pelas **anotações** do Spring Validation na classe do request:

```
@PostMapping
public ResponseEntity<Cliente> save(@RequestBody @Valid ClienteRequest request) {

          Cliente clienteRequisicao = request.buildCliente();
          StringBuilder erros = new StringBuilder();

          //O campo nome não pode ser nulo e nem vazio
          if (clienteRequisicao.getNome() == null ||
clienteRequisicao.getNome().equals("")) {
              erros.append("O campo Nome é de preenchimento obrigatório. ");
          }

          //O campo nome não pode ser maior que 100 caracteres
          ...


          //O campo fone tem que ter mais que 8 caracteres e menos que 20
          if (clienteRequisicao.getFone() != null &&
(clienteRequisicao.getFone().length() < 8 || clienteRequisicao.getFone().length() > 20))
{
              erros.append("O campo Fone tem que ter entre 8 e 20 caracteres. ");
          }

          if (erros.length() > 0) {
              throw new BadRequestException(erros.toString());
          }

          Cliente clienteSalvo = clienteService.save(clienteRequisicao);
          return new ResponseEntity<Cliente>(clienteSalvo, HttpStatus.CREATED);
}
```

```
package br.com.ifpe.oxefood.servicos.cliente;

import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;

import org.hibernate.validator.constraints.Length;
import org.hibernate.validator.constraints.br.CPF;

...
public class ClienteRequest {

   @NotNull(message = "O Nome é de preenchimento obrigatório")
   @NotEmpty(message = "O Nome é de preenchimento obrigatório")
   @Length(max = 100, message = "O Nome deverá ter no máximo {max} caracteres")
   private String nome;

   @JsonFormat(pattern = "dd/MM/yyyy")
   private LocalDate dataNascimento;

   @NotBlank(message = "O CPF é de preenchimento obrigatório")
   @CPF
   private String cpf;

   @Length(min = 8, max = 20, message = "O campo Fone tem que ter entre {min} e {max}
caracteres")
   private String foneCelular;

   private String foneFixo;

    ...
```

Validações no Back-end :: Entrada das Requisições

Atenção, para que as anotações do Spring Validation funcione, você precisa obrigatoriamente indicar no objeto recebido pelo endpoint do Controller a anotação **@Valid**:

```
@PostMapping
public ResponseEntity<Cliente> save(@RequestBody @Valid ClienteRequest request) {

	Cliente cliente = clienteService.save(request.build());
	return new ResponseEntity<Cliente>(cliente, HttpStatus.CREATED);
}
```

OBS.: Sem esta anotação, o Spring Validation não irá funcionar por mais que existam anotações na classe do request.

Validações no Back-end :: Entrada das Requisições

Anotações mais comuns do Spring Validation:

- `@NotNull` -> Válida se o campo está nulo.
- `@NotEmpty` -> Válida se o campo está vazio.
- `@NotBlank` -> Válida se o campo está nulo ou vazio.
- `@Past` -> Válida se a data é hoje ou qualquer dia antes de hoje.
- `@Future` -> Válida se a data é hoje ou qualquer dia depois de hoje.
- `@Length` -> Válida o tamanho mínimo e máximo de um campo.
- `@Max` -> Válida o tamanho máximo de um campo.
- `@Min` -> Valida o tamanho mínimo de um campo.
- `@Email` -> Verifica se o campo possui as características de um endereço de e-mail.

Há várias anotações que vieram para poder facilitar a nossa vida, como, por exemplo, `@Cpf`.

Validações no Banco de Dados

Validações no Back-end :: No Banco de Dados

A validação de dados é uma tarefa comum que ocorre em todas as camadas de um aplicativo, incluindo a persistência. O Java Persistence API (JPA) fornece suporte para a API de Validação de Bean (**Bean Validation**) para que a validação de dados possa ser feita no tempo de execução.

Bean Validation é uma especificação que permite validar objetos com facilidade em diferentes camadas da aplicação. A vantagem de usar Bean Validation é que as restrições ficam inseridas nas classes de modelo e, consequentemente, são reaplicadas em forma de constraints no banco de dados pelo JPA (framework que estamos utilizando para mapeamento objeto relacional).

Validações no Back-end :: No Banco de Dados

Temos como exemplo de implementação de validações com **Bean Validation** no código abaixo:

```
...
public class Cliente extends EntidadeAuditavel  {

   @OneToMany(mappedBy = "cliente", orphanRemoval = true, fetch = FetchType.EAGER)
   private List<EnderecoCliente> enderecos;

   @Column(nullable = false, length = 100)
   private String nome;

   @Column
   private LocalDate dataNascimento;

   @Column(unique = true)
   private String cpf;

   @Column
   private String foneCelular;

   @Column
   private String foneFixo;

}
```

O campo `nome` não poderá ser nulo e não pode pode ter mais que 100 caracteres ao ser salvo no banco de dados.

O campo `cpf` é único no banco de dados, ou seja, não poderá ser salvo um cliente com um CPF que já exista em algum outro cliente no banco de dados.

Validações de Regras de Negócio

Validações no Back-end :: De Regras de Negócio

Uma validação muito comum em projetos é verificar se o ID nas consultas por ID existem e retornar uma mensagem correta para o front-end.

```
public Cliente obterPorID(Long id) {

	return repository.findById(id).get();
}
```

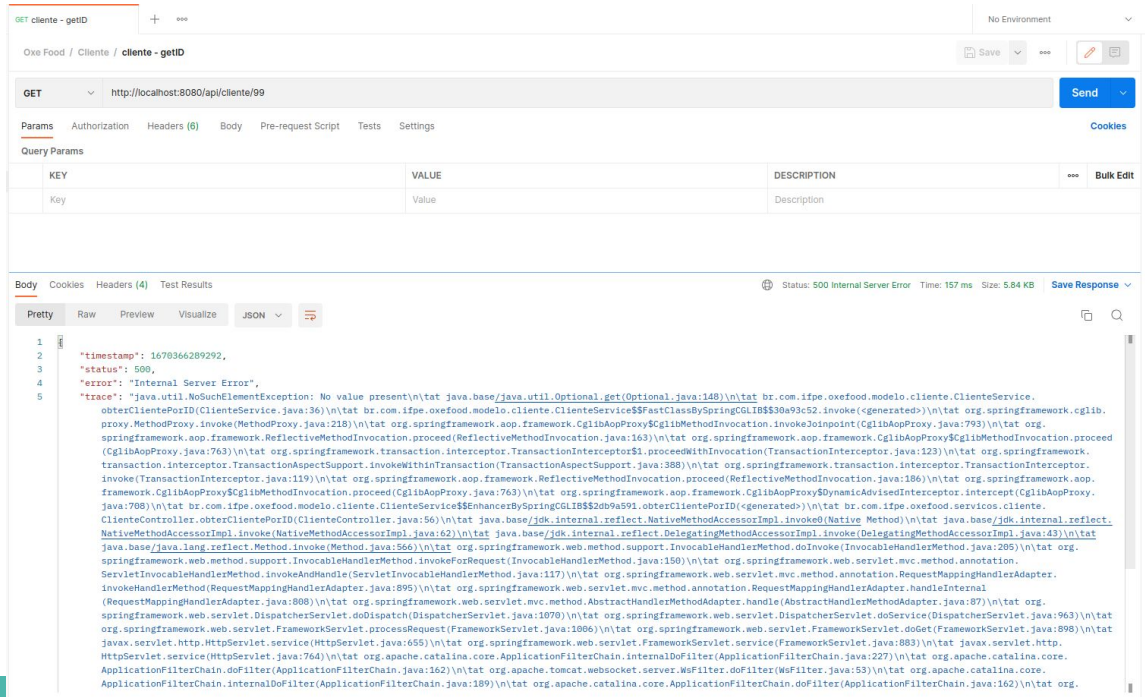
```
public Produto obterPorID(Long id) {

	return repository.findById(id).get();
}
```

```
public Entregador obterPorID(Long id) {

	return repository.findById(id).get();
}
```

Validações no Back-end :: De Regras de Negócio

Nos métodos findById implementados até o momento, caso fosse informado um ID inexistente um erro ilegível seria retornar para o front-end:

```
public Cliente obterPorID(Long id) {

	return repository.findById(id).get();
}
```



Validações no Back-end :: De Regras de Negócio

Desta forma, recomenda-se que seja feita a verificação através da classe **`java.util.Optional`**, e caso o `ID` não retorne um objeto na consulta, seja levantada uma exceção de negócio com uma mensagem customizada:

```
import java.util.Optional;
...
public Cliente obterPorID(Long id) {

      Optional<Cliente> consulta = repository.findById(id);

      if (consulta.isPresent()) {
          return consulta.get();
      } else {
          throw new EntidadeNaoEncontradaException("Cliente", id);
      }
}
```

```
package br.com.ifpe.oxefood.util.exception;

...
@ResponseStatus(code = HttpStatus.NOT_FOUND)
public class EntidadeNaoEncontradaException extends RuntimeException {

    public EntidadeNaoEncontradaException(String entidade, Long id) {
       super(String.format("Não foi encontrado(a) um(a) %s com o id %s", entidade, id.toString()));
    }
}
```

Validações no Back-end :: De Regras de Negócio

Você pode precisar também implementar alguma validação para alguma regra de negócio definida nos requisitos do sistema, por exemplo:

> No cadastro de Produtos, não permita que sejam inseridos projetos com valor abaixo de R$ 10.

Nesse caso, você precisará implementar uma validação específica a ser realizada na classe `Service` antes da operação ser realizada no banco de dados.

Validações no Back-end :: De Regras de Negócio

No cadastro de Produtos, não permita que sejam inseridos projetos com valor abaixo de R$ 10.

```
@Transactional
public Produto save(Produto produto) {

       if (produto.getValorUnitario() < 10) {
           throw new ProdutoException(ProdutoException.MSG_VALOR_MINIMO_PRODUTO);
       }

       produto.setHabilitado(Boolean.TRUE);
       produto.setVersao(1L);
       produto.setDataCriacao(LocalDate.now());
       return repository.save(produto);
}
```

```
@ResponseStatus(code = HttpStatus.INTERNAL_SERVER_ERROR)
public class ProdutoException extends RuntimeException {

    public static final String MSG_VALOR_MINIMO_PRODUTO = "Não é permitido inserir produtos com valores inferiores a R$ 10.";

    public ProdutoException(String msg) {

       super(String.format(msg));
    }
}
```

Dúvidas

Exercício

Implemente as validações abaixo nas funcionalidades de Categoria de Produto, Produto, Cliente, Entregador e Produto:

- Validações na Entrada das Requisições;
- Validações no Banco de Dados;
- Validações de Regras Negócio.
  - Não permitir que sejam inseridos clientes com um telefone que não tenha o prefixo 81
  - Não permitir que sejam inseridos produtos com um valor men
    que 100;

Obrigado !