

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

Validação e Testes (Tarefa B)

Prof. Fellipe Aleixo (fellipe.aleixo@ifrn.edu.br)

Iteração B1:

VALIDANDO OS DADOS

Feedback do Cliente

- Ao experimentar os resultados da primeira iteração, o usuário faz algumas observações:
 - “a aplicação não acusa nenhum erro ao se digitar um preço inválido, ou não preencher a descrição”
 - “você não poderia adicionar uma validação?”

Validação

- Onde devemos colocar?
 - O modelo é responsável pelo armazenamento das informações no banco de dados
- Vejamos o código da nossa classe de modelo...

```
class Product < ActiveRecord::Base  
end
```

Validação

- O método **validates()** é o método padrão para a criação de validadores em Rails
 - Responsável por checar um ou mais campos, de acordo com uma ou mais condições
 - Exemplo:

```
validates :title, :description, :image_url, presence: true
```

- Vejamos o que acontece quando tentamos inserir um produto sem um atributo com essa “restrição”

Validação



The screenshot shows a web browser window titled 'Depot' with the address bar displaying 'localhost:3000/products'. The page content is titled 'New product'. A red error message box at the top states: '3 errors prohibited this product from being saved:'. Below this, a list of errors is shown: 'Title can't be blank', 'Description can't be blank', and 'Image url can't be blank'. The form fields for 'Title', 'Description', and 'Image url' are highlighted with red borders, indicating they are the source of the errors. The 'Price' field is also visible but not highlighted. At the bottom of the form, there is a 'Create Product' button and a 'Back' link.

Depot

localhost:3000/products

New product

3 errors prohibited this product from being saved:

- Title can't be blank
- Description can't be blank
- Image url can't be blank

Title

Description

Image url

Price

Create Product

[Back](#)

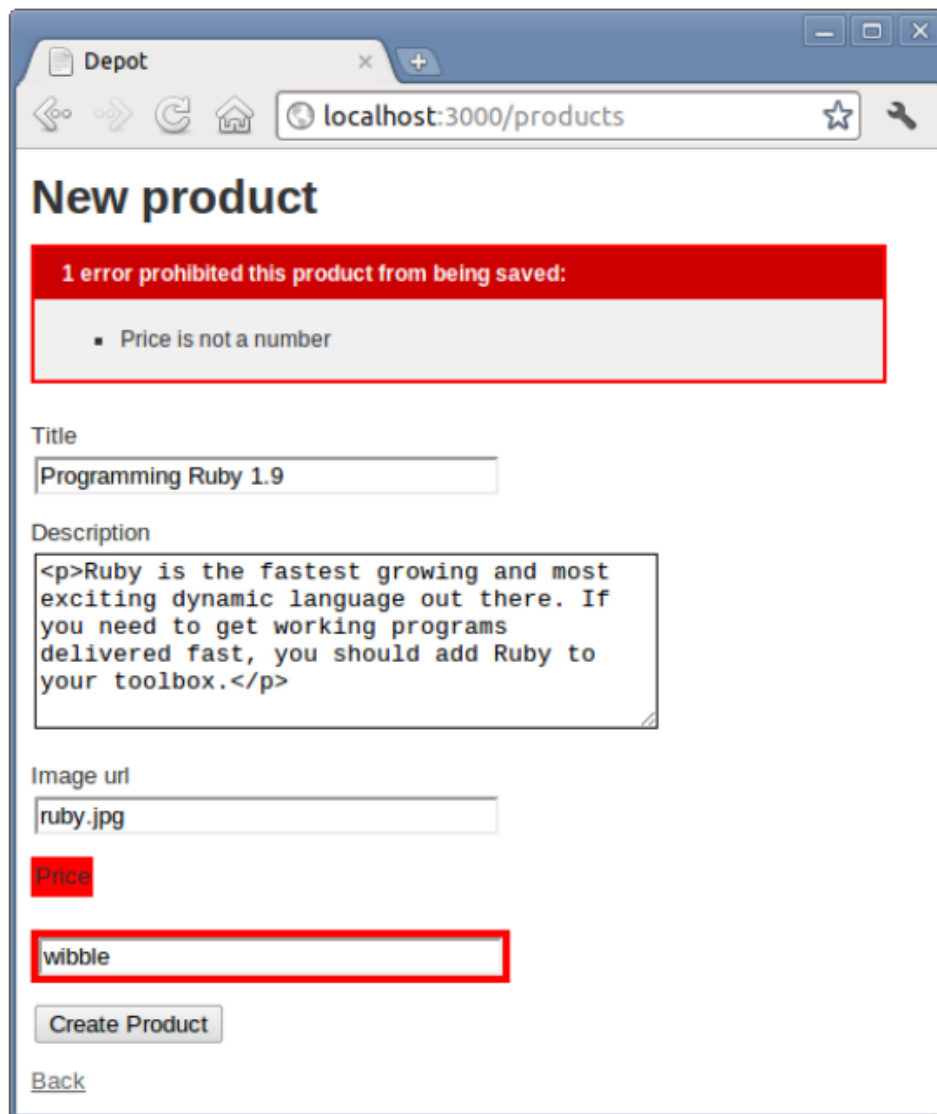
Validação

- Podemos checar também se o preço informado é válido (valor positivo)

```
validates :price, numericality: {greater_than_or_equal_to: 0.01}
```

- Vamos ver o que acontece quando não satisfazemos tal restrição...

Validação



The screenshot shows a web browser window with the address bar at `localhost:3000/products`. The page title is "New product". A red error message box states: "1 error prohibited this product from being saved:". Below this, a list of errors includes "Price is not a number". The form fields are: Title (text input with "Programming Ruby 1.9"), Description (text area with HTML content), Image url (text input with "ruby.jpg"), Price (text input with "wibble", highlighted with a red border), and a "Create Product" button. A "Back" link is at the bottom left.

Depot

localhost:3000/products

New product

1 error prohibited this product from being saved:

- Price is not a number

Title

Description

```
<p>Ruby is the fastest growing and most exciting dynamic language out there. If you need to get working programs delivered fast, you should add Ruby to your toolbox.</p>
```

Image url

Price

Create Product

[Back](#)

Validação

- Podemos querer garantir que cada produto tenha um título (ou nome) único

```
validates :title, uniqueness: true
```

- Podemos checar se um determinado campo satisfaz uma determinada expressão regular

```
validates :image_url, allow_blank: true, format: {  
  with: %r{\.(\.gif|jpg|png)\Z}i,  
  message: 'must be a URL for GIF, JPG or PNG image.'  
}
```

Validação

- Vejamos como ficou a nossa classe Produto:

Download rails40/depot_b/app/models/product.rb

```
class Product < ActiveRecord::Base
  validates :title, :description, :image_url, presence: true
  validates :price, numericality: {greater_than_or_equal_to: 0.01}
  validates :title, uniqueness: true
  validates :image_url, allow_blank: true, format: {
    with: %r{\.(gif|jpg|png)\Z}i,
    message: 'must be a URL for GIF, JPG or PNG image.'
  }
end
```

Testando a Aplicação

- Antes de prosseguir, testemos a aplicação

rake test

- ☹️ dessa vez foram identificadas algumas falhas
 - “should create product”
 - “should update product”
- Solução: criar dados de teste válidos

Testando a Aplicação

Download rails40/depot_b/test/controllers/products_controller_test.rb

```
require 'test_helper'
```

```
class ProductsControllerTest < ActionController::TestCase
```

```
  setup do
```

```
    @product = products(:one)
```

```
➤    @update = {
```

```
➤      title:      'Lorem Ipsum',
```

```
➤      description: 'Wibbles are fun!',
```

```
➤      image_url:  'lorem.jpg',
```

```
➤      price:      19.95
```

```
➤    }
```

```
  end
```

```
  test "should get index" do
```

```
    get :index
```

```
    assert_response :success
```

```
    assert_not_nil assigns(:products)
```

```
  end
```

Testando a Aplicação

```
test "should get new" do
  get :new
  assert_response :success
end
```

```
test "should create product" do
  assert_difference('Product.count') do
    ➤ post :create, product: @update
  end

  assert_redirected_to product_path(assigns(:product))
end
```

```
# ...
test "should update product" do
  ➤ patch :update, id: @product, product: @update
  assert_redirected_to product_path(assigns(:product))
end
```

```
# ...
end
```

Iteração B2:

TESTE UNITÁRIO DO MODELO

Testes Unitários

- O Rails cria a infraestrutura de teste no momento em que cria a aplicação
 - Exemplo:

```
depot> ls test/models
product_test.rb
```
 - Estimula a criação dos testes o quanto antes

Testes Unitários

- Vejamos a classe de teste que foi gerada

[Download rails40/depot_a/test/models/product_test.rb](#)

```
require 'test_helper'
```

```
class ProductTest < ActiveSupport::TestCase
  # test "the truth" do
  #   assert true
  # end
end
```


Testes Unitários

- Qual o objetivo?
 - Verificar a presença de erros
- Como fazer isso?
 - Testando se as saídas dos métodos são as esperadas
 - Indicar ao framework de teste quando o código “passa” ou “falha”
 - Uso de asserções – *assertions* – método **assert()**

Testes Unitários

Download rails40/depot_b/test/models/product_test.rb

```
test "product attributes must not be empty" do
  product = Product.new
  assert product.invalid?
  assert product.errors[:title].any?
  assert product.errors[:description].any?
  assert product.errors[:price].any?
  assert product.errors[:image_url].any?
end
```

– Métodos **errors()** e **invalid?()** da classe de modelo

Testes Unitários

- Executando apenas os testes das classes de modelo

```
depot> rake test:models
```

```
.  
Finished tests in 0.257961s, 3.8766 tests/s, 19.3828 assertions/s.  
1 tests, 5 assertions, 0 failures, 0 errors, 0 skips
```

Testes Unitários

- Podemos checar (antes) se as validações funcionarão como definido

[Download rails40/depot_c/test/models/product_test.rb](#)

```
test "product price must be positive" do
  product = Product.new(title:      "My Book Title",
                        description: "yyy",
                        image_url:   "zzz.jpg")

  product.price = -1
  assert product.invalid?
  assert_equal ["must be greater than or equal to 0.01"],
    product.errors[:price]

  product.price = 0
  assert product.invalid?
  assert_equal ["must be greater than or equal to 0.01"],
    product.errors[:price]

  product.price = 1
  assert product.valid?
end
```

Ambientes para Executar Testes

- Provê a infraestrutura (e os recursos) para que os testes sejam executados
 - Especificação de um “conteúdo inicial” das classes de modelo sendo testadas
 - Podem ser especificadas no diretório **test/fixtures**
 - Contendo dados no formato YAML
 - Por exemplo: o arquivo **products.yml**

Ambientes para Executar Testes

```
Download rails40/depot_b/test/fixtures/products.yml
```

```
# Read about fixtures at
```

```
# http://api.rubyonrails.org/classes/ActiveRecord/Fixtures.html
```

```
one:
```

```
  title: MyString
```

```
  description: MyText
```

```
  image_url: MyString
```

```
  price: 9.99
```

```
two:
```

```
  title: MyString
```

```
  description: MyText
```

```
  image_url: MyString
```

```
  price: 9.99
```

Ambientes para Executar Testes

Download rails40/depot_c/test/fixtures/products.yml

ruby:

title: *Programming Ruby 1.9*

description:

Ruby is the fastest growing and most exciting dynamic language out there. If you need to get working programs delivered fast, you should add Ruby to your toolbox.

price: *49.50*

image_url: *ruby.png*

```
class ProductTest < ActiveSupport::TestCase
  ➤ fixtures :products
    #...
end
```