PMR 3304 - Sistemas de Informação

Laboratório Aula 01

Prof. Marcos S. G. Tsuzuki e José Reinaldo Silva

PMR - EPUSP

23 de agosto de 2019

Criando Aplicação de Manutenção de Produtos

Durante as primeiras aulas de laboratório, será feito o desenvolvimento incremental de uma aplicação. O sistema será especificado minimamente e já iniciaremos o seu desenvolvimento.

Será desenvolvida uma aplicação que possui um catálogo de produtos que podem ser visualizados. Os produtos são selecionados e incluídos em um carrinho de compras, será possível ir e vir do catálogo de produtos para o carrinho de compras. Uma vez que as compras foram finalizadas, o usuário poderá realizar o checkout, exibindo um sumários dos itens selecionados e finalizando com o pagamento.

Uma vez que a descrição foi feita, uma nova aplicação **depot** será criada. Para que cada aluno tenha a sua versão, mesmo em outras turmas, o diretório deverá conter o seu número USP, não coloque os símbolos < e >, apenas o número USP direto após depot.

```
rails new depot<NUSP>
```

Em seguida, avance para o diretório recém criado.

```
cd depot<NUSP>
```

Será utilizado o banco de dados SQLite 3.0. Este é o banco de dados padrão para o desenvolvimento Rails e foi instalado juntamento com o Rails. A primeira etapa será executar o **scaffold** considerando que o produto possui um conjunto básico de atributos como: título, descrição, imagem e preço. Cada atributo possui um tipo: o título é uma string, a descrição é um texto, e assim por diante.

```
rails generate scaffold Product title:string description:text
                                image_url:string price:decimal
```

Este comando criou diversos arquivos, sendo que um especial é o arquivo de migration que foi armazenado no diretório **db/migrate** com nome ***_create_products.rb**. Um migration representa uma modificação a ser feita no banco de dados. O arquivo atual já espelha a configuração atual, mas vamos realizar uma pequena modificação. Abra o arquivo de migration utilizando o NotePad++[1], ele possui a configuração abaixo:

```
class CreateProducts < ActiveRecord::Migration[5.1]
  def change
    create_table :products do |t|
      t.string :title
      t.text :description
      t.string :image_url
      t.decimal :price

```

[1] https://notepad-plus-plus.org

```
9       t.timestamps
10    end
11  end
12 end
```

Vamos modificar o preço para que fique com oito digitos e duas casa decimais. As modificações ocorreram na linha 7 da listagem abaixo.

```
1  class CreateProducts < ActiveRecord::Migration[5.1]
2    def change
3      create_table :products do |t|
4        t.string :title
5        t.text :description
6        t.string :image_url
7        t.decimal :price, precision: 8, scale: 2
8
9        t.timestamps
10     end
11   end
12 end
```

Para tornar estas mudanças efetivas, é necessário executar o comando migrate, como abaixo:

```
1 rails db:migrate
```

Vamos observar o funcionamento da aplicação recém criada. Inicialmente, o servidor será iniciado:

```
1 rails server
```

Este servidor utiliza a porta 3000. Se você obter um erro indicando que o endereço já está em uso quando o servidor for iniciado, significa que você já possui um servidor Rails executando em sua máquina. Para interromper a execução do servidor utilize *Ctrl-C*. Inicie um browser e insira a URL `http://localhost:3000/products`.

Caso ocorra um erro na execução, no arquivo **app/view/layouts/application.html.erb** nas linhas 7 e 8 da listagem abaixo, tente substituir a linha 8 por **javascript_pack_tag** ao invés de **javascript_include_tag**. Caso ainda persista o erro, é porque o Node.js [2] não foi instalado ou porque o computador não foi reiniciado após a instalação.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Depot</title>
5     <%= csrf_meta_tags %>
```

[2] `http:/nodejs.org`.

```
6
7       <%= stylesheet_link_tag    'application', media: 'all', 'data-
            turbolinks-track': 'reload' %>
8       <%= javascript_include_tag 'application', 'data-turbolinks-track': '
            reload' %>
9     </head>
10
11    <body>
12      <%= yield %>
13    </body>
14  </html>
```

Agora, os erros devem ter sido eliminados. Continuando a execução, este exemplo exibe uma lista vazia de produtos.

Primeira Atividade

Como primeira atividade, vamos modificar o número de linhas e colunas do campo "*description*". Para este fim, abra o arquivo **app/views/products/_form.html.erb**, que possui o conteúdo abaixo:

```
1  <%= form_with(model: product, local: true) do |form| %>
2    <% if product.errors.any? %>
3      <div id="error_explanation">
4        <h2><%= pluralize(product.errors.count, "error") %> prohibited
             this product from being saved:</h2>
5
6        <ul>
7        <% product.errors.full_messages.each do |message| %>
8          <li><%= message %></li>
9        <% end %>
10       </ul>
11     </div>
12   <% end %>
13
14   <div class="field">
15     <%= form.label :title %>
16     <%= form.text_field :title %>
17   </div>
18
19   <div class="field">
20     <%= form.label :description %>
21     <%= form.text_area :description %>
22   </div>
23
24   <div class="field">
```

```
25      <%= form.label :image_url %>
26      <%= form.text_field :image_url %>
27    </div>
28
29    <div class="field">
30      <%= form.label :price %>
31      <%= form.text_field :price %>
32    </div>
33
34    <div class="actions">
35      <%= form.submit %>
36    </div>
37  <% end %>
```

Na linha 21, vamos acrescentar o código **, rows: 10, cols: 60**, conforme a listagem abaixo.

```
1   <%= form_with(model: product, local: true) do |form| %>
2     <% if product.errors.any? %>
3       <div id="error_explanation">
4         <h2><%= pluralize(product.errors.count, "error") %> prohibited
              this product from being saved:</h2>
5
6         <ul>
7         <% product.errors.full_messages.each do |message| %>
8           <li><%= message %></li>
9         <% end %>
10        </ul>
11      </div>
12    <% end %>
13
14    <div class="field">
15      <%= form.label :title %>
16      <%= form.text_field :title %>
17    </div>
18
19    <div class="field">
20      <%= form.label :description %>
21      <%= form.text_area :description, rows:10, cols: 60 %>
22    </div>
23
24    <div class="field">
25      <%= form.label :image_url %>
26      <%= form.text_field :image_url %>
27    </div>
28
29    <div class="field">
30      <%= form.label :price %>
31      <%= form.text_field :price %>
32    </div>
```

```
33
34    <div class="actions">
35      <%= form.submit %>
36    </div>
37  <% end %>
```

Continue a execução no browser e clique em *New Produto*. Em seguida, forneça os dados para o cadastro de um produto e clique em *Create Produto*.Ao retornar, o produto inserido deve aparecer na listagem. Agora, será necessário parar o servidor pressionando *Ctrl-C*.

Caminhemos para o ambiente de testes, e execute o primeiro comando neste ambiente:

```
1 rails test
```

Possivelmente surgirá uma requisição para executar o migration em ambiente de testes.

```
1 rails db:migrate RAILS_ENV=test
```

Agora, execute o ambiente de testes novamente. O resultado deve informar que zero erros estão presentes.

Tornando as Listagens mais Bonitas

O povoamento do banco de dados poder ser feito pela inserção de dados pelo formulário recém contruído, o que não é muito prático. O Rails disponibiliza uma ferramenta para este fim. Vamos editar o arquivo **db/seeds.rb**. Mas, ao invés de editá-lo, vamos recuperar um arquivo pronto existente no pacote disponibilizado no moodle, substitua o arquivo recuperado pelo existente em seu projeto.

Aproveite que você está recuperando arquivos e coloque as imagens disponilizadas no diretório **app/assets/images**.

Abaixo, é fornecida uma listagem do arquivo **db/seeds.rb** recuperado.

```
1  #---
2  #---
3  # encoding: utf-8
4  Product.delete_all
5  Product.create!(title: 'Autorretrato e Outras Cronicas',
6    description:
7      %{<p>
8        <em>Autorretrato e Outras Cronicas</em>
9        Em edicao comemorativa dos 75 anos do Grupo Editorial Record, esta
10           coletanea de cronicas de Carlos Drummond de Andrade, com selecao de
11           Fernando Py, e relancada com capa nova e encarte contendo fotos e
12           uma selecao de correspondencias ineditas do poeta com Alfredo Machado,
13           fundador do Grupo. Com o habitual sentimento terno e amargo diante
14           dos absurdos e da beleza da vida, Drummond registra, com perfeicao
15           e elegancia, fatos da vida diaria e da politica brasileira, a morte
16           de amigos, a natureza, a literatura, e ate um bem-humorado e razoavelmente
```

```
17            condescendente autorretrato: "O sr. Carlos Drummond de Andrade e um razoavel
18            prosador que se julga bom poeta, no que se ilude. Como prosador, assinou
19            algumas cronicas e alguns contos que revelam certo conhecimento das formas
20            graciosas de expressao, certo humour e malicia."
21            </p>},
22    imagem_url: 'Livro1.jpg',
23    price: 45.00)
24  # . . .
25  Product.create!(title: 'Uma Forma de Saudade. Paginas de Diario',
26    description:
27      %{<p>
28        <em>Uma Forma de Saudade. Paginas de Diario</em>
29        Trinta anos apos a morte de Carlos Drummond de Andrade, as paginas arrancadas
30            de seu diario e guardadas por sua filha Maria Julieta num envelope com a
31            inscricao "Diario de papai/ Familia e amigos" compoem Uma forma de saudade,
32            edicao especial que a Companhia das Letras lancara em 31 de outubro deste
33            ano, Dia D. Precedidas por uma introducao de Pedro Augusto Grana Drummond,
34            organizador do volume, e sucedidas por uma seleta da poemas, as paginas
35            revelam ao publico reflexoes do poeta acerca de familiares e amigos proximos
36            como Manuel Bandeira e Rodrigo Melo Franco de Andrade. O livro conta ainda
37            com fotos do arquivo da familia e fac-similes das anotacoes do poeta, alem
38            de projeto grafico especial, assinado por Raul Loureiro.
39        </p>},
40    imagem_url: 'Livro2.jpg',
41    price: 26.00)
42  # . . .
43
44  Product.create!(title: 'Declaracao de Amor',
45    description:
46      %{<p>
47        <em>Declaracao de Amor</em>
48        Com o subtitulo "Cancao de namorados", esta reuniao de poemas amorosos,
49            romanticos e deliciosamente apaixonados de Carlos Drummond de Andrade
50            mostra a faceta mais lirica do grande poeta mineiro. Textos ja classicos
51            ou que merecem uma nova leitura, como "Amar", "Lembrete", "Ausencia",
52            "Toada do amor", "Declaracao de amor" e "O chao e cama" foram criteriosamente
53            selecionados por Luis Mauricio e Pedro Augusto Grana Drummond, netos do poeta
54            e grandes conhecedores de sua obra. O resultado e uma celebracao de beijos,
55            abracos e carinhos - uma festa para o amor, enfim. Com projeto grafico exclusivo
56            e ilustracoes do aclamado artista Nik Neves, Declaracao de amor e o presente
57            ideal para o Dia dos Namorados.
58        </p>},
59    imagem_url: 'Livro3.jpg',
60    price: 46.00)
```

Agora, execute o comando para povoar o banco de dados:

```
1  rails db:seed
```

Vamos melhorar o visual do site incluindo algum CSS. O arquivo **app/assets/stylesheets/-products.scss** será modificado para:

```
1  // Place all the styles related to the Products controller here.
2  // They will automatically be included in application.css.
3  // You can use Sass (SCSS) here: http://sass-lang.com/
4  // Place all the styles related to the Products controller here.
5  // They will automatically be included in application.css.
6  // You can use Sass (SCSS) here: http://sass-lang.com/
```

```
7  .products {
8    table {
9      border-collapse: collapse;
10   }
11
12   table tr td {
13     padding: 5px;
14     vertical-align: top;
15   }
16
17   .list_image {
18     width:  60px;
19     height: 70px;
20   }
21
22   .list_description {
23     width: 60%;
24
25     dl {
26       margin: 0;
27     }
28
29     dt {
30       color:        #244;
31       font-weight:  bold;
32       font-size:    larger;
33     }
34
35     dd {
36       margin: 0;
37     }
38   }
39
40   .list_actions {
41     font-size:    x-small;
42     text-align:   right;
43     padding-left: 1em;
44   }
45
46   .list_line_even {
47     background:   #e0f8f8;
48   }
49
50   .list_line_odd {
51     background:   #f8b0f8;
52   }
53 }
```

O arquivo **app/view/layouts/application.html.erb** contendo a definição de layout para

toda a aplicação (já apresentando anteriormente) será modificado na linha 11, para:

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Depot</title>
5      <%= csrf_meta_tags %>
6
7      <%= stylesheet_link_tag    'application', media: 'all', 'data-
          turbolinks-track': 'reload' %>
8      <%= javascript_include_tag 'application', 'data-turbolinks-track': '
          reload' %>
9    </head>
10
11   <body class='<%= controller.controller_name %>'>
12     <%= yield %>
13   </body>
14 </html>
```

O arquivo **app/views/products/index.html.erb** contendo a view atual está listado abaixo:

```
1  <p id="notice"><%= notice %></p>
2
3  <h1>Products</h1>
4
5  <table class="products">
6    <thead>
7      <tr>
8        <th>Title</th>
9        <th>Description</th>
10       <th>Image url</th>
11       <th>Price</th>
12       <th colspan="3"></th>
13     </tr>
14   </thead>
15
16   <tbody>
17     <% @products.each do |product| %>
18       <tr>
19         <td><%= product.title %></td>
20         <td><%= product.description %></td>
21         <td><%= product.image_url %></td>
22         <td><%= product.price %></td>
23         <td><%= link_to 'Show', product %></td>
24         <td><%= link_to 'Edit', edit_product_path(product) %></td>
25         <td><%= link_to 'Destroy', product, method: :delete, data: {
              confirm: 'Are you sure?' } %></td>
26       </tr>
27     <% end %>
```

```
28   </tbody>
29 </table>
30
31 <br>
32
33 <%= link_to 'New Product', new_product_path %>
```

Observe que exibe uma lista de produtos, e para cada item é possível exibir detalhes, modificar e destruir. Também está disponível um link para criação de um novo produto na parte inferior. Vamos criar um novo arquivo **app/views/products/index.html.erb** que substituirá este e se conectará diretamente ao arquivo **products.scss** recém criado. A listagem do novo arquivo está abaixo:

```
1  <p id="notice"><%= notice %></p>
2
3  <h1>Products</h1>
4
5  <table>
6  <% @products.each do |product| %>
7    <tr class="<%= cycle('list_line_odd', 'list_line_even') %>">
8
9      <td>
10       <%= image_tag(product.image_url, class: 'list_image') %>
11     </td>
12
13     <td class="list_description">
14       <dl>
15         <dt><%= product.title %></dt>
16         <dd><%= truncate(strip_tags(product.description),
17                 length: 80) %></dd>
18       </dl>
19     </td>
20
21     <td class="list_actions">
22       <%= link_to 'Show', product %><br/>
23       <%= link_to 'Edit', edit_product_path(product) %><br/>
24       <%= link_to 'Destroy', product, method: :delete,
25                   data: { confirm: 'Are you sure?' } %>
26     </td>
27   </tr>
28 <% end %>
29 </table>
30
31 <br />
32
33 <%= link_to 'New product', new_product_path %>
```

Execute novamente e verifique como ficou no browser. Faça vários testes neste ambiente, edite, tente remover, e outros. Ao final, aborte a execução do servidor.

Agora vamos armazenar o estado atual do código no **git**.

```
1 git init
2 git add .
3 git commit -m "Aula01 Parte1"
```

Validação

Vamos trabalhar no arquivo de modelo **app/models/product.rb**:

```
1 class Product < ApplicationRecord
2 end
```

A primeira validação indicará que nenhum dos campos (título, descrição, e url da imagem) devem ser fornecidos. Modifique o arquivo para que fique como o arquivo abaixo.

```
1 class Product < ApplicationRecord
2   validates :title, :description, :image_url, presence: true
3
4 end
```

Agora, tente inserir um produto totalmente em branco, e verifique as mensagens de erro. Em seguida, vamos inserir novas validações para o preço que deverá ser um valor numérico maior ou igual a $0,01$. Vamos incluir também uma verificação para que o título do produto seja único. E por último vamos validar para que a URL fornecida para uma imagem seja válida, verificando se a URL temina com **.gif**, **.jpg**

Vamos verificar também se o preço é um valor válido, positivo. Será utilizada uma opção *numerically()* para realizar a verificação. Inclua a linha abaixo no arquivo de modelo **app/models/product.rb**:

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

Agora tente inserir um valor não válido e veja a mensagem de erro exibida.

Uma outra opção será que todos os títulos sejam únicos, ou seja, não existam títulos de livro repetidos.

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

Finalmente, será necessário validar que a URL fornecida para a imagem seja válida. A opção de arquivo deve terminar com **,gif**, **.jpg** ou **.png**. Observer que a opção *allow_ blank* foi utilizada, evitando múltiplas mensagens de erro quando o campo estiver vazio.

```
  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.'
  }
```

O seu modelo de Product deve ter ficado como:

```
class Product < ApplicationRecord
  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
```

Teste novamente a sua aplicação e verifique que agora ela está mais robusta. Erros aparecem se os campos não forem fornecidos corretamente. Ao final aborte a execução do servidor.

Testes

Vamos realizar novamente o ambiente de testes.

```
rails test
```

Agora é possível observar que erros aparecem. Um em *should create product* e outro em *should update product*. O ideal é que seja fornecido um dado completo em **test/controllers/-products_controller_test.rb**. O arquivo deve ser modificado para inserir a definição do produto como na linha 6 **@update**, e as linhas 26 e 43 também devem ser modificadas para ficar como na listagem.

```
require 'test_helper'

class ProductsControllerTest < ActionDispatch::IntegrationTest
  setup do
    @product = products(:one)
    @update = {
      title:       'Lorem Ipsum',
      description: 'Wibbles are fun!',
      image_url:   'lorem.jpg',
      price:       19.95
    }
```

```
12    end
13
14    test "should get index" do
15      get products_url
16      assert_response :success
17    end
18
19    test "should get new" do
20      get new_product_url
21      assert_response :success
22    end
23
24    test "should create product" do
25      assert_difference('Product.count') do
26        post products_url, params: { product: @update }
27      end
28
29      assert_redirected_to product_url(Product.last)
30    end
31
32    test "should show product" do
33      get product_url(@product)
34      assert_response :success
35    end
36
37    test "should get edit" do
38      get edit_product_url(@product)
39      assert_response :success
40    end
41
42    test "should update product" do
43      patch product_url(@product), params: { product: @update}
44      assert_redirected_to product_url(@product)
45    end
46
47    test "should destroy product" do
48      assert_difference('Product.count', -1) do
49        delete product_url(@product)
50      end
51
52      assert_redirected_to products_url
53    end
54  end
```

Após realizar as mudanças, execute novamente os testes e verifique que o relatório indica que tudo ocorreu bem. Algumas versões podem surgir problemas pois o banco de dados de teste não foi povoado, assim a verificação da linha 14 (*should get index*) pode gerar falha.

Testando o Modelo

É possível testar apenas o modelo, verificando os subdiretórios identifica-se o arquivo **test/models/product_test.rb**.

```
1 require 'test_helper'
2
3 class ProductTest < ActiveSupport::TestCase
4   # test "the truth" do
5   #   assert true
6   # end
7 end
```

Esta listagem apenas identifica que *ProductTest* é uma subclasse de *ActiveSupport::TestCase*. Vamos caminhar para criar um ambiente de testes verdadeiro, será criado um Product e em seguida os campos criados serão verificados e devem estar vazios. Este teste está apenas confirmando isto.

```
1 require 'test_helper'
2
3 class ProductTest < ActiveSupport::TestCase
4   test "product attributes must not be empty" do
5     product = Product.new
6     assert product.invalid?
7     assert product.errors[:title].any?
8     assert product.errors[:description].any?
9     assert product.errors[:price].any?
10    assert product.errors[:image_url].any?
11  end
12 end
```

Faça os testes com o seguinte comando:

```
1 rails test:models
```

Vamos adicionar um novo teste que verifica se o preço é um valor positivo. Nos dois primeiros casos, o valor do preço é inválido e recebemos a informação de valor incorreto. Enquanto que apenas no terceiro, o valor é correto. Execute os testes e confirme que está tudo em ordem como previsto.

```
1   test "product price must be positive" do
2     product = Product.new(title:       "My Book Title",
3                           description: "yyy",
4                           image_url:   "zzz.jpg")
5     product.price = -1
```

```
6     assert product.invalid?
7     assert_equal ["must be greater than or equal to 0.01"],
8       product.errors[:price]
9
10    product.price = 0
11    assert product.invalid?
12    assert_equal ["must be greater than or equal to 0.01"],
13      product.errors[:price]
14
15    product.price = 1
16    assert product.valid?
17  end
```

Vamos agora verificar o terminador do nome do arquivo de imagem. Foram criadas duas listas, uma contendo nomes aceitos pelo sistema, e outro com nomes que não são aceitos pelo sistema. Execute os testes e confirme que está tudo em ordem como previsto.

```
1   def new_product(image_url)
2     Product.new(title:       "My Book Title",
3                 description: "yyy",
4                 price:       1,
5                 image_url:   image_url)
6   end
7
8   test "image url" do
9     ok = %w{ fred.gif fred.jpg fred.png FRED.JPG FRED.Jpg
10              http://a.b.c/x/y/z/fred.gif }
11    bad = %w{ fred.doc fred.gif/more fred.gif.more }
12
13    ok.each do |name|
14      assert new_product(name).valid?, "#{name} shouldn't be invalid"
15    end
16
17    bad.each do |name|
18      assert new_product(name).invalid?, "#{name} shouldn't be valid"
19    end
20  end
```

Finalmente, vamos agora criar um teste para confirmar que o nome é único (sem repetições). Para isto, vamos utilizar uma outra ferramenta conhecida como *Fixtures*. Neste caso, os dados estão no formato YAML. Cada arquivo contém dados um único modelo. O nome do *Fixture* é relevante: o nome base do arquivo precisa casar com o nome da tabela do banco de dados. Neste caso, ele será chamado como **products.yml**, e o Rails já criou o arquivo para nós:

```
1 # Read about fixtures at
2 # http://api.rubyonrails.org/classes/ActiveRecord/FixtureSet.html
3
4 one:
```

```
5   title: MyString
6   description: MyText
7   image_url: MyString
8   price: 9.99
9
10 two:
11  title: MyString
12  description: MyText
13  image_url: MyString
14  price: 9.99
```

Vamos acrescentar a este arquivo um nome relevante para o nosso propósito.

```
1  # Read about fixtures at
2  # http://api.rubyonrails.org/classes/ActiveRecord/FixtureSet.html
3
4  one:
5    title: MyString
6    description: MyText
7    image_url: MyString
8    price: 9.99
9
10 two:
11   title: MyString
12   description: MyText
13   image_url: MyString
14   price: 9.99
15
16 ruby:
17   title: Programming Ruby 1.9
18   description:
19     Ruby is the fastest growing and most exciting dynamic
20     language out there. If you need to get working programs
21     delivered fast, you should add Ruby to your toolbox.
22   image_url: Ruby.png
23   price: 49.50
```

Vamos modificar o arquivo **test/models/product_test.rb**. Em algumas versões é necessário incluir o comando para execução da *Fixture* logo no início da classe. Em outras, o carregamento é automático. A chamada **products(:ruby)** retorna o modelo do produto contido nos dados definidos na *Fixture*. O teste assume que já existe um registro para o livro Ruby, e recupera o título pela chamada **products(:ruby).title**, e cria um novo modelo do produto com o título e outros parâmetros. Ocorre um erro ao tentar o salvamento do registro, e a mensagem de erro está associada ao atributo **title**.

```
1  test "product is not valid without a unique title" do
2     product = Product.new(title:       products(:ruby).title,
3                           description: "yyy",
```

```
4                             price:       1,
5                             image_url:   "fred.gif")
6
7     assert product.invalid?
8     assert_equal ["has already been taken"], product.errors[:title]
9   end
```

Vamos verificar com o git, quais arquivos diferem da última versão.

```
1 git status
```

Como apenas modificamos arquivos existentes e não acrescentamos novos, é possível combinar **git commit** e **git add** em um único comando.

```
1 git commit -a -m 'Validation'
```