PMR 3304 - Sistemas de Informação

Laboratório Aula 02

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

PMR - EPUSP

23 de agosto de 2019

Criando Aplicação para Exibição de Catálogos

Na aula de laboratório, foi criado um **controller** para produtos, que será utilizado em nossa aplicação pelo **adminitrador**. Agora, vamos criar um segundo **controller** que interagirá com o pagamento de clientes que selecionarão os produtos. Vamos chamá-lo de **store**.

```
1 rails generate controller Store index
```

Este comando solicitou que o rails crie a classe **StoreController** em **store_controller.rb** contendo um único método **index**. Vamos executar o servidor e executar a aplicação no browser com a URL `http://localhost:3000/store/index`.

Vamos melhorar isto, modificando o arquivo de roteamento **config/routes.rb**. Que deverá ficar como (inserir a linha 2):

```
1 Rails.application.routes.draw do
2   root 'store#index', as: 'store_index'
3
4   get 'store/index'
5   resources :products
6   # For details on the DSL available within this file, see http://guides
       .rubyonrails.org/routing.html
7 end
```

Assim, fizemos uma configuração para que o acesso **root** vá diretamente para o **controller** que criamos. Verique o funcionamento com a URL `http://localhost:3000`.

Exibindo Produtos

Vamos exibir uma lista de produtos a partir do banco de dados. Para isto vamos modificar o método **index** em **store_controller.rb** (inserir a linha 3):

```
1 class StoreController < ApplicationController
2   def index
3     @products = Product.order(:title)
4   end
5 end
```

Este comando exibirá os produtos em ordem alfabética. Vamos modificar a view template presente em **app/views/store/index.html.erb**.

```
1 <p id="notice"><%= notice %></p>
2
3 <h1>Your Catalog</h1>
4
```

```
5  <% @products.each do |product| %>
6    <div class="store">
7      <div class="entry">
8        <%= image_tag(product.image_url) %>
9        <h3><%= product.title %></h3>
10       <%= sanitize(product.description) %>
11       <div class="price_line">
12         <span class="price"><%= product.price %></span>
13       </div>
14     </div>
15   </div>
16 <% end %>
```

A utilização de **sanitize** faz com que o HTML seja gerado com estilos de modo seguro. Observe que alguns caracteres foram removidos em relação à aula passada. Também foi utilizado o método **helper image_tag**. Este método cria uma tag HTML <img> e utiliza como argumento a origem da imagem.

Verique o funcionamento com a URL `http://localhost:3000`.

Vamos acrescentar algum estilo ao catálogo. Para isto utilize o arquivo fornecido em "Arquivos Complementares II"**store.scss. Este arquivo deve ser colocado no diretório app/assets/stylesheets.**

```
1  // Place all the styles related to the Store controller here.
2  // They will automatically be included in application.css.
3  // You can use Sass (SCSS) here: http://sass-lang.com/
4  .store {
5    h1 {
6      margin: 0;
7      padding-bottom: 0.5em;
8      font:  150% sans-serif;
9      color: #226;
10     border-bottom: 3px dotted #77d;
11   }
12
13   /* An entry in the store catalog */
14   .entry {
15     overflow: auto;
16     margin-top: 1em;
17     border-bottom: 1px dotted #77d;
18     min-height: 100px;
19
20     img {
21       width: 80px;
22       margin-right: 5px;
23       margin-bottom: 5px;
24       position: absolute;
25     }
```

```
26
27      h3 {
28        font-size: 120%;
29        font-family: sans-serif;
30        margin-left: 100px;
31        margin-top: 0;
32        margin-bottom: 2px;
33        color: #227;
34      }
35
36      p, div.price_line {
37        margin-left: 100px;
38        margin-top: 0.5em;
39        margin-bottom: 0.8em;
40      }
41
42      .price {
43        color: #44a;
44        font-weight: bold;
45        margin-right: 3em;
46      }
47    }
48  }
```

Verique o funcionamento com a URL `http://localhost:3000`**.**

Acrescentando Layout de Página

As páginas em um site, geralmente compartilham um layout, um designer utiliza um template padrão que será utilizado em todo o site. Ele fica localizado em application.html.erb. Inicialmente, vamos criar uma barra lateral. O arquivo deverá ficar como (modificar a linha 4 e inserir o conteúdo entre as linhas 15 e 28, também inserir as linhas 30 e 31):

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Books Online Store</title>
5      <%= csrf_meta_tags %>
6
7      <%= csrf_meta_tags %>
8
9      <%= stylesheet_link_tag    'application', media: 'all',
10     'data-turbolinks-track': 'reload' %>
11     <%= javascript_include_tag 'application', 'data-turbolinks-track': '
          reload' %>
12   </head>
```

```
13
14    <body class="<%= controller.controller_name %>">
15      <div id="banner">
16        <%= image_tag 'logo.svg', alt: 'The Pragmatic Bookshelf' %>
17        <span class="title"><%= @page_title %></span>
18      </div>
19      <div id="columns">
20        <div id="side">
21          <ul>
22            <li><a href="http://www....">Home</a></li>
23            <li><a href="http://www..../faq">Questions</a></li>
24            <li><a href="http://www..../news">News</a></li>
25            <li><a href="http://www..../contact">Contact</a></li>
26          </ul>
27        </div>
28        <div id="main">
29          <%= yield %>
30        </div>
31      </div>
32    </body>
33  </html>
```

Este código faz acesso ao arquivo logo.svg que foi fornecido junto com o pacote. ele deve ser colocado no diretório app/assets/images. Este formato de arquivo pode ser editado com o Inkscape [1] software gratuito.

Verique o funcionamento com a URL `http://localhost:3000`**.**

Veja que ele não funcionou corretamente, pois existe a necessidade de modificarmos o stylesheet, arquivo app/assets/stylesheets/aplication.css. Este arquivo precisa ser renomeado para application.scss. O conteúdo do arquivo foi fornecido no primeiro arquivo complementar no site da disciplina.

```
1  /*
2   * This is a manifest file that'll be compiled into application.css,
       which will
3   * include all the files listed below.
4   *
5   * Any CSS and SCSS file within this directory, lib/assets/stylesheets,
6   * vendor/assets/stylesheets, or any plugin's vendor/assets/stylesheets
7   * directory can be referenced here using a relative path.
8   *
9   * You're free to add application-wide styles to this file and they'll
       appear
10  * at the bottom of the compiled file so the styles you add here take
11  * precedence over styles defined in any other CSS/SCSS files in this
```

[1] https://inkscape.org/en/

```
12  * directory. Styles in this file should be added after the last require
       _*
13  * statement. It is generally better to create a new file per style
       scope.
14  *
15  *= require_tree .
16  *= require_self
17  */
18 body, body > p, body > ol, body > ul, body > td {margin: 8px !important}
19
20 #banner {
21   position: relative;
22   min-height: 40px;
23   background: #9c9;
24   padding: 10px;
25   border-bottom: 2px solid;
26   font: small-caps 40px/40px "Times New Roman", serif;
27   color: #282;
28   text-align: center;
29
30   img {
31     position: absolute;
32     top: 0;
33     left: 0;
34     width: 192px;
35   }
36 }
37
38 #notice {
39   color: #000 !important;
40   border: 2px solid red;
41   padding: 1em;
42   margin-bottom: 2em;
43   background-color: #f0f0f0;
44   font: bold smaller sans-serif;
45 }
46
47 #notice:empty {
48   display: none;
49 }
50
51 #columns {
52   background: #141;
53   display: flex;
54
55   #main {
56     padding: 1em;
57     background: white;
58     flex: 1;
```

```
59    }
60
61    #side {
62      padding: 1em 2em;
63      background: #141;
64
65      ul {
66        padding: 0;
67
68        li {
69          list-style: none;
70
71          a {
72            color: #bfb;
73            font-size: small;
74          }
75        }
76      }
77    }
78  }
79
80  @media all and (max-width: 800px) {
81    #columns {
82      flex-direction: column-reverse;
83    }
84  }
85
86  @media all and (max-width: 500px) {
87    #banner {
88      height: 1em;
89    }
90
91    #banner .title {
92      display: none;
93    }
94  }
```

Como explicado nos comentários do arquivo, o arquivo manifest incluirá automaticamente todos os stylesheets disponíveis no diretório e em qualquer subdiretório. Isto é feito pela diretiva require_tree.

Este arquivo contém três principais áreas: o banner (no topo), a área principal e a área lateral à esquerda. Cada uma destas áreas possui margens, fontes, e cores. O banner é centrado e indica que a imagem deve ser localizada à esquerda. O stylesheet também inclui acomodação para dispositivos móveis, que deevem possuir telas menores. Tente reduzir a dimensão da janela e veja o que ocorre!

Utilizando Helper para Formatar o Preço

O ruby fornece um método para formatação e que pode ser utilizado para formatar o preço. Este método pode ser colocado diretamente na view app/views/store/index.html.erb:

```
1       <span class="price">
2         <%= sprintf("$%0.02f", product.price) %>
3       </span>
```

Testes

Vamos continuar com alguns testes, você já deve ter um arquivo de fixture products.yml, que fica no diretório depot/test/fixture. Verifique se o seu arquivo possui o conteúdo abaixo:

```
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: ruby.png
8    price: 9.99
9
10 two:
11   title: MyString
12   description: MyText
13   image_url: ruby.png
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   price:       49.50
23   image_url:   ruby.png
```

Execute o comando abaixo:

```
1  rails test
```

Se ocorreu algum erro, possivelmente o arquivo ruby.png não existe e deve ser colocado em app/assets/images. Vamos observar o que o rails gerou para nós no arquivo depot/test/controller/store_controller_test.rb:

```
1 require 'test_helper'
2
3 class StoreControllerTest < ActionDispatch::IntegrationTest
4   test "should get index" do
5     get store_index_url
6     assert_response :success
7   end
8
9 end
```

o teste "should get index" assegura uma resposta segura. Mas, também desejamos que a resposta contenha o nosso layout, a informação do produto, e a formatação do preço. Veja como ele deve ficar:

```
1 require 'test_helper'
2
3 class StoreControllerTest < ActionDispatch::IntegrationTest
4   test "should get index" do
5     get store_index_url
6     assert_response :success
7     assert_select '#columns #side a', minimum: 4
8     assert_select '#main .entry', 3
9     assert_select 'h3', 'Programming Ruby 1.9'
10    assert_select '.price', /\$[,\d]+\.\d\d/
11  end
12
13 end
```

As linhas 7 a 10 observam o HTML retornado, utilizando notação CSS. Apenas como referência, as seleções que iniciam com o caracter "#" indicam id de atributos, seleções que iniciam com "." indicam atributos de classe, e seleções que não contém prefixo indicam nomes dos elementos.

Assim, a primeira seleção procura por um elemento de nome a que está contido em um id de nome side, que está contido em um elemento com id e com o valor de columns. Este teste verifica que um mínimo de quatro destes elementos existem. As três linhas seguintes asseguram que todos os produtos são exibidos. O primeiro verifica que três elementos com a classe de nome entry são internos a p arte main da página. A próxima verifica que existe um elemento h3 com o título Programming Ruby 1.9. A terceira linha verifica se o preço está formatado corretamente.

Execute o comando abaixo:

```
rails test:controllers
```

Veja que os testes ocorreram sem erro, exceto pelos mesmos que já existiam anteriormente.

Caching Resultados Parciais

Se tudo ocorrer como planejado, possivelmente o site será muito acessado. Assim que a página for solicitada, os itens serão recuperados do banco e visualizados. Entretanto, como os dados do banco não devem mudar muito, podemos melhorar o acesso com *caching*. Para isto vamos configurar o ambiente:

```
rails dev:cache
```

Vamos marcar as seções de nosso template (em depot/app/view/store) que precisam de atualização se algum produto for modificado, e dentro destas seções marcamos as subseções que serão necessárias para a atualização de um produto específico que foi modificado. veja nas linhas 5 e 7.

```
1  <p id="notice"><%= notice %></p>
2
3  <h1>Your Catalog</h1>
4
5  <% cache @products do %>
6    <% @products.each do |product| %>
7      <% cache product do %>
8        <div class="entry">
9          <%= image_tag(product.image_url) %>
10         <h3><%= product.title %></h3>
11         <%= sanitize(product.description) %>
12         <div class="price_line">
13           <span class="price"><%= number_to_currency(product.price) %></
                span>
14         </div>
15       </div>
16     <% end %>
17   <% end %>
18 <% end %>
```

Para que você perceba o funcionamento, será necessário que você observe o servidor em execução do prompt de comando. Pressione o botão "F5" forçando a recarga da tela pelo browser. Se o servidor deverá exibir apenas um *Read Fragment*

Criação do Carrinho

Com a exibição do catálogo, a próxima etapa é a criação de um carrinho de compras. O usuário seleciona produtos que deseja virtualmente e os coloca em seu carrinho de compras. A primeira etapa nesta direção é a criação de uma entidade para conter os itens acrescentados. Portanto, um carrinho será armazenado no banco de dados e ele possuirá um identificador que ficará armazenado na sessão do usuário.

```
rails generate scaffold Cart
```

Vamos criar o arquivo app/controller/concerns/current_cart.rb conforme descrito abaixo:

```
module CurrentCart

  private

    def set_cart
      @cart = Cart.find(session[:cart_id])
    rescue ActiveRecord::RecordNotFound
      @cart = Cart.create
      session[:cart_id] = @cart.id
    end
end
```

O método set_cart inicia recuperando o :cart_id do objeto session e em seguida recupera o cart correspondente a este *id*. Se o cart não for encontrado, será criado um novo cart e o *id* criado será armazenado na session.

Conectando Produtos ao Carrinho

O cart conterá os produtos, assim vamos criar a conexão entre os produtos e o cart.

```
rails generate scaffold LineItem product:references cart:belongs_to
```

Em seguida vamos executar o migrate:

```
rails db:migrate
```

O banco de dados possui agora um local para armazenar as referências entre itens, carts e produtos. Vamos observar o conteúdo de app/models/line_item.rb.

```
1 class LineItem < ApplicationRecord
2   belongs_to :product
3   belongs_to :cart
4 end
```

Segundo o modelo isto significa que um item possui uma referência para produto e outra referência para o cart. Agora precisamos modificar o modelo do cart para inserir esta representação.

```
1 class Cart < ApplicationRecord
2   has_many :line_items, dependent: :destroy
3 end
```

Isto representa que um cart possui muitos items. Ou seja, podemos colocar em um carrinho diversos itens. A palavra dependent indica que a existência do item depende do cart. Se o cart for destruído, então os itens também devem ser destruídos. Da mesma maneira, o modelo de produtos também deve ser modificado para (ver linhas 2, 4 e entre 20 e 26):

```
1  class Product < ApplicationRecord
2    has_many :line_items
3
4    before_destroy :ensure_not_referenced_by_any_line_item
5
6    #...
7
8    validates :title, :description, :image_url, presence: true
9    validates :price, numericality: {greater_than_or_equal_to: 0.01}
10 #
11   validates :title, uniqueness: true
12   validates :image_url, allow_blank: true, format: {
13     with:    %r{\.(gif|jpg|png)\Z}i,
14     message: 'must be a URL for GIF, JPG or PNG image.'
15   }
16   validates :title, length: {minimum: 10}
17
18   private
19
20     # ensure that there are no line items referencing this product
21     def ensure_not_referenced_by_any_line_item
22       unless line_items.empty?
23         errors.add(:base, 'Line Items present')
24         throw :abort
25       end
26     end
27 end
```

Aqui é definido um método que verifica se um usuário inseriu o produto em um cart. Neste caso, o produto não poderá ser removido pois o item do cart se torna inválido.

Antes de seguirmos vamos incluir um teste que garante que um produto presente em um cart não pode ser removido (código entre as linhas 47 e 53).

```
1  require 'test_helper'
2
3  class ProductsControllerTest < ActionDispatch::IntegrationTest
4    setup do
5      @product = products(:one)
6      @update = {
7        title:       'Lorem Ipsum',
8        description: 'Wibbles are fun!',
9        image_url:   'lorem.jpg',
10       price:       19.95
11     }
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 "can't delete product in cart" do
48      assert_difference('Product.count', 0) do
49        delete product_url(products(:two))
50      end
51
52      assert_redirected_to products_url
53    end
54
55    test "should destroy product" do
56      assert_difference('Product.count', -1) do
57        delete product_url(@product)
58      end
59
60      assert_redirected_to products_url
61    end
62  end
```

O arquivo test/fixtures/line_items.yml também precisa ser modificado para garantir que o produto *two* está presente am ambos os carts.

```
1  # Read about fixtures at http://api.rubyonrails.org/classes/ActiveRecord
      /FixtureSet.html
2
3  one:
4    product: two
5    cart: one
6
7  two:
8    product: two
9    cart: two
```

Ainda não executaremos os testes. Vamos aguardar um pouco.

Acrescentando um Botão para Incluir Produtos ao Cart

Não é necessário criar um novo *controller* ou mesmo uma *action* para este fim. Observe as *actions* criadas pelo *scaffold generator*: index(), show(), new(), edit(), create(), update() e destroy(). Todos estão no arquivo app/controllers/line_items_controller.rb. O mais apropriado será o método create() utilizando o método HTTP Post. Como será criado um botão, vamos fazer uso do método button_to(). Cada produto possuirá um botão e conterá o identificador do produto. Vamos modificar a view app/view/store/index.html.erb (veja a linha 14).

```
<p id="notice"><%= notice %></p>

<h1>Your Catalog</h1>

<% cache @products do %>
  <% @products.each do |product| %>
    <% cache product do %>
      <div class="entry">
        <%= image_tag(product.image_url) %>
        <h3><%= product.title %></h3>
        <%= sanitize(product.description) %>
        <div class="price_line">
          <span class="price"><%= number_to_currency(product.price) %></
             span>
          <%= button_to 'Add to Cart', line_items_path(product_id:
             product) %>
        </div>
      </div>
    <% end %>
  <% end %>
<% end %>
```

Falta apenas corrigir uma formatação para que o botão fique alinhado com o preço (veja as linhas entre 37 e 44).

```
// Place all the styles related to the Store controller here.
// They will automatically be included in application.css.
// You can use Sass (SCSS) here: http://sass-lang.com/
.store {
  h1 {
    margin: 0;
    padding-bottom: 0.5em;
    font:  150% sans-serif;
    color: #226;
    border-bottom: 3px dotted #77d;
  }

  /* An entry in the store catalog */
  .entry {
    overflow: auto;
    margin-top: 1em;
    border-bottom: 1px dotted #77d;
    min-height: 100px;

    img {
      width: 80px;
      margin-right: 5px;
      margin-bottom: 5px;
      position: absolute;
```

```
25      }
26
27      h3 {
28        font-size: 120%;
29        font-family: sans-serif;
30        margin-left: 100px;
31        margin-top: 0;
32        margin-bottom: 2px;
33        color: #227;
34      }
35
36  //
37      p, div.price_line {
38        margin-left: 100px;
39        margin-top: 0.5em;
40        margin-bottom: 0.8em;
41        form, div {
42          display: inline;
43        }
44      }
45  //
46
47      .price {
48        color: #44a;
49        font-weight: bold;
50        margin-right: 3em;
51      }
52    }
53  }
```

Verifique o funcionamento do site executando o servidor e observando no browser.

Vamos agora implementar a inserção de um item no carrinho, inserindo um código no arquivo app/controller/line_items_controller.rb (veja as linhas 2, 3, 29, 30 e 34).

```
1  class LineItemsController < ApplicationController
2    include CurrentCart
3    before_action :set_cart, only: [:create]
4    before_action :set_line_item, only: [:show, :edit, :update, :destroy]
5
6    # GET /line_items
7    # GET /line_items.json
8    def index
9      @line_items = LineItem.all
10   end
11
12   # GET /line_items/1
13   # GET /line_items/1.json
14   def show
```

```
15   end
16
17   # GET /line_items/new
18   def new
19     @line_item = LineItem.new
20   end
21
22   # GET /line_items/1/edit
23   def edit
24   end
25
26   # POST /line_items
27   # POST /line_items.json
28   def create
29     product = Product.find(params[:product_id])
30     @line_item = @cart.line_items.build(product: product)
31
32     respond_to do |format|
33       if @line_item.save
34         format.html { redirect_to @line_item.cart,
35           notice: 'Line item was successfully created.' }
36         format.json { render :show,
37           status: :created, location: @line_item }
38       else
39         format.html { render :new }
40         format.json { render json: @line_item.errors,
41           status: :unprocessable_entity }
42       end
43     end
44   end
45
46   # PATCH/PUT /line_items/1
47   # PATCH/PUT /line_items/1.json
48   def update
49     respond_to do |format|
50       if @line_item.update(line_item_params)
51         format.html { redirect_to @line_item, notice: 'Line item was
               successfully updated.' }
52         format.json { render :show, status: :ok, location: @line_item }
53       else
54         format.html { render :edit }
55         format.json { render json: @line_item.errors, status: :
               unprocessable_entity }
56       end
57     end
58   end
59
60   # DELETE /line_items/1
61   # DELETE /line_items/1.json
```

```
62    def destroy
63      @line_item.destroy
64      respond_to do |format|
65        format.html { redirect_to line_items_url, notice: 'Line item was
              successfully destroyed.' }
66        format.json { head :no_content }
67      end
68    end
69
70    private
71      # Use callbacks to share common setup or constraints between actions
          .
72      def set_line_item
73        @line_item = LineItem.find(params[:id])
74      end
75
76      # Never trust parameters from the scary internet, only allow the
           white list through.
77      def line_item_params
78        params.require(:line_item).permit(:product_id, :cart_id)
79      end
80    #...
81  end
```

Assim, ao se criar um item, será procurado o cart onde vamos incluir o item. Se ele não existir, será criado. O método create tmbém foi modificado para que os parâmetros fossem inseridos corretamente: cart e produto. O inserir um item, aparecerá uma mensagem informando que o item foi inserido. Como foi criada a *view* para o cart e não foi fornecido nenhum atributo. Vamos modificar o arquivo app/views/carts/show.html.erb para termos algum resutado. O arquivo é completamente novo.

```
1  <p id="notice"><%= notice %></p>
2
3  <h2>Your Cart</h2>
4  <ul>
5    <% @cart.line_items.each do |item| %>
6      <li><%= item.product.title %></li>
7    <% end %>
8  </ul>
```

Execute o servidor e teste o código desenvolvido. Insira diversos produtos, inclusive alguns do mesmo livro. Veja o que o mesmo livro aparecerá múltiplas vezes.

O arquivo de testes test/controllers/line_items_controller_test.rb precisa ser modificado para (veja linhas 20, 23 a 26):

```
1  require 'test_helper'
2
3  class LineItemsControllerTest < ActionDispatch::IntegrationTest
4    setup do
5      @line_item = line_items(:one)
6    end
7
8    test "should get index" do
9      get line_items_url
10     assert_response :success
11   end
12
13   test "should get new" do
14     get new_line_item_url
15     assert_response :success
16   end
17
18   test "should create line_item" do
19     assert_difference('LineItem.count') do
20       post line_items_url, params: { product_id: products(:ruby).id }
21     end
22
23     follow_redirect!
24
25     assert_select 'h2', 'Your Cart'
26     assert_select 'li', 'Programming Ruby 1.9'
27   end
28
29   test "should show line_item" do
30     get line_item_url(@line_item)
31     assert_response :success
32   end
33
34   test "should get edit" do
35     get edit_line_item_url(@line_item)
36     assert_response :success
37   end
38
39   test "should update line_item" do
40     patch line_item_url(@line_item), params: { line_item: { cart_id:
          @line_item.cart_id, product_id: @line_item.product_id } }
41     assert_redirected_to line_item_url(@line_item)
42   end
43
44   test "should destroy line_item" do
45     assert_difference('LineItem.count', -1) do
46       delete line_item_url(@line_item)
47     end
48
```

```
49      assert_redirected_to line_items_url
50    end
51  end
```

Vamos executar os testes (possivelmente será solicitado para executar o migrate). Estamos passando por meio de *post* o *id* de um produto criado a partir de :ruby. Estamos redirecionando para o cart, onde o *id* do cart está armazenado em um *session*. Assim, testa-se o que o usuário verá: o produto carregado e o título da página.

```
1  rails test test/controllers/line_items_controller_test.rb
```

Adicionando Quantidades ao Item

```
1  rails generate migration add_quantity_to_line_items quantity:integer
```

O rails deduzirá que você está acrescentando uma coluna à tabela line_items. Vamos modificar o arquivo de migration criado incluindo um valor padrão (veja a linha 3):

```
1  class AddQuantityToLineItems < ActiveRecord::Migration[5.2]
2    def change
3      add_column :line_items, :quantity, :integer, default: 1
4    end
5  end
```

Agora com isto pronto, vamos executar o migration:

```
1  rails db:migrate
```

Vamos precisar de um método para add_product inteligente para o cart em app/-models/cart.rb. Caso o produto já esteja na lista, a quantidade será incrementada. E caso ele ainda não esteja na lista, ele será incluído com uma unidade (veja as linhas 4 a 12).

```
1  class Cart < ApplicationRecord
2    has_many :line_items, dependent: :destroy
3
4    def add_product(product)
5      current_item = line_items.find_by(product_id: product.id)
6      if current_item
7        current_item.quantity += 1
8      else
```

```
9         current_item = line_items.build(product_id: product.id)
10      end
11      current_item
12    end
13  end
```

O controller app/controller/line_items_controller.rb também deverá ser modificado para utilizar este método (veja a linha 30, ela deve ser modificada).

```
1  class LineItemsController < ApplicationController
2    include CurrentCart
3    before_action :set_cart, only: [:create]
4    before_action :set_line_item, only: [:show, :edit, :update, :destroy]
5
6    # GET /line_items
7    # GET /line_items.json
8    def index
9      @line_items = LineItem.all
10   end
11
12   # GET /line_items/1
13   # GET /line_items/1.json
14   def show
15   end
16
17   # GET /line_items/new
18   def new
19     @line_item = LineItem.new
20   end
21
22   # GET /line_items/1/edit
23   def edit
24   end
25
26   # POST /line_items
27   # POST /line_items.json
28   def create
29     product = Product.find(params[:product_id])
30     @line_item = @cart.add_product(product)
31
32     respond_to do |format|
33       if @line_item.save
34         format.html { redirect_to @line_item.cart,
35           notice: 'Line item was successfully created.' }
36         format.json { render :show,
37           status: :created, location: @line_item }
38       else
39         format.html { render :new }
40         format.json { render json: @line_item.errors,
```

```
41            status: :unprocessable_entity }
42        end
43      end
44    end
45
46    # PATCH/PUT /line_items/1
47    # PATCH/PUT /line_items/1.json
48    def update
49      respond_to do |format|
50        if @line_item.update(line_item_params)
51          format.html { redirect_to @line_item, notice: 'Line item was
                successfully updated.' }
52          format.json { render :show, status: :ok, location: @line_item }
53        else
54          format.html { render :edit }
55          format.json { render json: @line_item.errors, status: :
                unprocessable_entity }
56        end
57      end
58    end
59
60    # DELETE /line_items/1
61    # DELETE /line_items/1.json
62    def destroy
63      @line_item.destroy
64      respond_to do |format|
65        format.html { redirect_to line_items_url, notice: 'Line item was
              successfully destroyed.' }
66        format.json { head :no_content }
67      end
68    end
69
70    private
71      # Use callbacks to share common setup or constraints between actions
            .
72      def set_line_item
73        @line_item = LineItem.find(params[:id])
74      end
75
76      # Never trust parameters from the scary internet, only allow the
            white list through.
77      def line_item_params
78        params.require(:line_item).permit(:product_id, :cart_id)
79      end
80    #...
81  end
```

A view app/views/cars/show.html.erb precisará ser modificada para exibir este campo (veja a linha 6).

```
<p id="notice"><%= notice %></p>

<h2>Your Cart</h2>
<ul>
  <% @cart.line_items.each do |item| %>
    <li><%= item.quantity %> &times; <%= item.product.title %></li>
  <% end %>
</ul>
```

Tudo pronto, vamos executar o servidor e visualizar no browser. Insira novos produtos no cart. Possivelmente, alguns produtos aparecem repetidos e outros com quantidades maiores que 1. Vamos criar uma migration que compactará o banco de dados, sem necessidade de zerá-lo.

```
rails generate migration combine_items_in_cart
```

Neste caso, os métodos serão totalmente novos, e vamos criar um método up e outro down, um o inverso do outro (isto é necessário pois temos a possibilidade de executar o rollback. O arquivo de migration deverá ficar como:

```
class CombineItemsInCart < ActiveRecord::Migration[5.2]
  def up
    # replace multiple items for a single product in a cart with a
    # single item
    Cart.all.each do |cart|
      # count the number of each product in the cart
      sums = cart.line_items.group(:product_id).sum(:quantity)

      sums.each do |product_id, quantity|
        if quantity > 1
          # remove individual items
          cart.line_items.where(product_id: product_id).delete_all

          # replace with a single item
          item = cart.line_items.build(product_id: product_id)
          item.quantity = quantity
          item.save!
        end
      end
    end
  end

  def down
    # split items with quantity>1 into multiple items
    LineItem.where("quantity>1").each do |line_item|
      # add individual items
      line_item.quantity.times do
        LineItem.create(
```

```
29            cart_id: line_item.cart_id,
30            product_id: line_item.product_id,
31            quantity: 1
32          )
33        end
34
35        # remove original item
36        line_item.destroy
37      end
38    end
39  end
```

Ao executar o migrate, todos os dados serão compactados.

```
1  rails db:migrate
```

O método de testes também deverá ser reescrito, pois o cart exibe a quandidade de itens agora, O test/controllers/line_items_controller_test.rb ficará (veja a linha 26) como abaixo. O código apresentado é a *escape sequence* para o caracter especial.

```
1  require 'test_helper'
2
3  class LineItemsControllerTest < ActionDispatch::IntegrationTest
4    setup do
5      @line_item = line_items(:one)
6    end
7
8    test "should get index" do
9      get line_items_url
10     assert_response :success
11   end
12
13   test "should get new" do
14     get new_line_item_url
15     assert_response :success
16   end
17
18   test "should create line_item" do
19     assert_difference('LineItem.count') do
20       post line_items_url, params: { product_id: products(:ruby).id }
21     end
22
23     follow_redirect!
24
25     assert_select 'h2', 'Your Cart'
26     assert_select 'li', "1 \u00D7 Programming Ruby 1.9"
27   end
28
```

```
29   test "should show line_item" do
30     get line_item_url(@line_item)
31     assert_response :success
32   end
33
34   test "should get edit" do
35     get edit_line_item_url(@line_item)
36     assert_response :success
37   end
38
39   test "should update line_item" do
40     patch line_item_url(@line_item), params: { line_item: { cart_id:
         @line_item.cart_id, product_id: @line_item.product_id } }
41     assert_redirected_to line_item_url(@line_item)
42   end
43
44   test "should destroy line_item" do
45     assert_difference('LineItem.count', -1) do
46       delete line_item_url(@line_item)
47     end
48
49     assert_redirected_to line_items_url
50   end
51 end
```

Vamos executar novamente os testes (talvez seja solicitada a execução do migrate em ambiente de testes).

```
1 rails test test/controllers/line_items_controller_test.rb
```

Processando Erros

Observe que durante a execução, a URL apareceu com o índice do cart. O que acontece se inserirmos um cart não existente como `https://localhost:3000/carts/wibble`. Veja que neste caso aparece uma mensagem de erro. É uma tentativa de entrar no sistema sem ser pelo processo convencional. Vamos inserir um código no controller app/controllers/carts_controller.rb para que este erro seja processado. Vamos utilizar uma estrutura conhecida como *flash*, onde é possível armazenar informações para recuperação posterior (veja as linhas 3, 78 a 81). Também modificaremos a destoruição do cart, é necessário que o cart destruído seja o seu (veja as linhas 59 a 61).

```
1 class CartsController < ApplicationController
2   before_action :set_cart, only: [:show, :edit, :update, :destroy]
3   rescue_from ActiveRecord::RecordNotFound, with: :invalid_cart
4   # GET /carts
```

```
5   # GET /carts.json
6   def index
7     @carts = Cart.all
8   end
9
10  # GET /carts/1
11  # GET /carts/1.json
12  def show
13  end
14
15  # GET /carts/new
16  def new
17    @cart = Cart.new
18  end
19
20  # GET /carts/1/edit
21  def edit
22  end
23
24  # POST /carts
25  # POST /carts.json
26  def create
27    @cart = Cart.new(cart_params)
28
29    respond_to do |format|
30      if @cart.save
31        format.html { redirect_to @cart, notice: 'Cart was successfully
              created.' }
32        format.json { render :show, status: :created, location: @cart }
33      else
34        format.html { render :new }
35        format.json { render json: @cart.errors, status: :unprocessable_
              entity }
36      end
37    end
38  end
39
40  # PATCH/PUT /carts/1
41  # PATCH/PUT /carts/1.json
42  def update
43    respond_to do |format|
44      if @cart.update(cart_params)
45        format.html { redirect_to @cart, notice: 'Cart was successfully
              updated.' }
46        format.json { render :show, status: :ok, location: @cart }
47      else
48        format.html { render :edit }
49        format.json { render json: @cart.errors, status: :unprocessable_
              entity }
```

```
50        end
51      end
52    end
53
54    # DELETE /carts/1
55    # DELETE /carts/1.json
56    def destroy
57      @cart.destroy if @cart.id == session[:cart_id]
58      session[:cart_id] = nil
59      respond_to do |format|
60        format.html { redirect_to store_index_url,
61          notice: 'Your cart is currently empty' }
62        format.json { head :no_content }
63      end
64    end
65
66    # ...
67    private
68    # ...
69
70      def set_cart
71        @cart = Cart.find(params[:id])
72      end
73
74      # Never trust parameters from the scary internet, only allow the
          white list through.
75      def cart_params
76        params.fetch(:cart, {})
77      end
78      def invalid_cart
79        logger.error "Attempt to access invalid cart #{params[:id]}"
80        redirect_to store_index_url, notice: 'Invalid cart'
81      end
82  end
```

A cláusula rescue_from intercepta a exceção gerada pelo cart.find(). O método invalid_cart utilizar o logger do rails para armazenar a mensagem de erro. Em seguida, redireciona para mostrar a loja com redirect_to(). Agora com este código, tente executar o código incorreto novamente.

Mesmo assim, temos uma outra preocupação. Para evitarmos de vez os parâmetros pela URL é possível modificar o controller app/controller/line_items_controller.rb (veja as linhas 78 a 80).

```
1  class LineItemsController < ApplicationController
2    include CurrentCart
3    before_action :set_cart, only: [:create]
4    before_action :set_line_item, only: [:show, :edit, :update, :destroy]
5
```

```
6   # GET /line_items
7   # GET /line_items.json
8   def index
9     @line_items = LineItem.all
10  end
11
12  # GET /line_items/1
13  # GET /line_items/1.json
14  def show
15  end
16
17  # GET /line_items/new
18  def new
19    @line_item = LineItem.new
20  end
21
22  # GET /line_items/1/edit
23  def edit
24  end
25
26  # POST /line_items
27  # POST /line_items.json
28  def create
29    product = Product.find(params[:product_id])
30    @line_item = @cart.add_product(product)
31
32    respond_to do |format|
33      if @line_item.save
34        format.html { redirect_to @line_item.cart,
35          notice: 'Line item was successfully created.' }
36        format.json { render :show,
37          status: :created, location: @line_item }
38      else
39        format.html { render :new }
40        format.json { render json: @line_item.errors,
41          status: :unprocessable_entity }
42      end
43    end
44  end
45
46  # PATCH/PUT /line_items/1
47  # PATCH/PUT /line_items/1.json
48  def update
49    respond_to do |format|
50      if @line_item.update(line_item_params)
51        format.html { redirect_to @line_item, notice: 'Line item was
              successfully updated.' }
52        format.json { render :show, status: :ok, location: @line_item }
53      else
```

```
54          format.html { render :edit }
55          format.json { render json: @line_item.errors, status: :
               unprocessable_entity }
56        end
57      end
58    end
59
60    # DELETE /line_items/1
61    # DELETE /line_items/1.json
62    def destroy
63      @line_item.destroy
64      respond_to do |format|
65        format.html { redirect_to line_items_url, notice: 'Line item was
             successfully destroyed.' }
66        format.json { head :no_content }
67      end
68    end
69
70    private
71      # Use callbacks to share common setup or constraints between actions
          .
72      def set_line_item
73        @line_item = LineItem.find(params[:id])
74      end
75
76      # Never trust parameters from the scary internet, only allow the
           white
77      # list through.
78      def line_item_params
79        params.require(:line_item).permit(:product_id)
80      end
81    #...
82  end
```

Os teste para apagar o cart em app/test/controllers/carts_controller_test.rb devem ser modificados devido ao uso da *session* (veja entre as linhas 42 e 49).

```
1  require 'test_helper'
2
3  class CartsControllerTest < ActionDispatch::IntegrationTest
4    setup do
5      @cart = carts(:one)
6    end
7
8    test "should get index" do
9      get carts_url
10     assert_response :success
11   end
12
```

```
13   test "should get new" do
14     get new_cart_url
15     assert_response :success
16   end
17
18   test "should create cart" do
19     assert_difference('Cart.count') do
20       post carts_url, params: { cart: {  } }
21     end
22
23     assert_redirected_to cart_url(Cart.last)
24   end
25
26   test "should show cart" do
27     get cart_url(@cart)
28     assert_response :success
29   end
30
31   test "should get edit" do
32     get edit_cart_url(@cart)
33     assert_response :success
34   end
35
36   test "should update cart" do
37     patch cart_url(@cart), params: { cart: {  } }
38     assert_redirected_to cart_url(@cart)
39   end
40
41   test "should destroy cart" do
42     post line_items_url, params: { product_id: products(:ruby).id }
43     @cart = Cart.find(session[:cart_id])
44
45     assert_difference('Cart.count', -1) do
46       delete cart_url(@cart)
47     end
48
49     assert_redirected_to store_index_url
50   end
51 end
```

Execute os testes agora, devem ser livre de erros.

```
1 rails test:controllers
```

Finalizando o Cart

Para a finalização do cart, é adequado que seja criado um botão para zerá-lo. Assim todos os produtos serão removidos. Vamos inicialmente criar o botão em app/views/cart/show.html.erb (veja as linhas 10 e 11).

```
1  <p id="notice"><%= notice %></p>
2
3  <h2>Your Cart</h2>
4  <ul>
5    <% @cart.line_items.each do |item| %>
6      <li><%= item.quantity %> &times; <%= item.product.title %></li>
7    <% end %>
8  </ul>
9
10 <%= button_to 'Empty cart', @cart, method: :delete,
11     data: { confirm: 'Are you sure?' } %>
```

Vamos remover a mensagem de que o "*Line item was successfully created*" em app/controllers/line_items_controller.rb (veja a linha 34).

```
1  class LineItemsController < ApplicationController
2    include CurrentCart
3    before_action :set_cart, only: [:create]
4    before_action :set_line_item, only: [:show, :edit, :update, :destroy]
5
6    # GET /line_items
7    # GET /line_items.json
8    def index
9      @line_items = LineItem.all
10   end
11
12   # GET /line_items/1
13   # GET /line_items/1.json
14   def show
15   end
16
17   # GET /line_items/new
18   def new
19     @line_item = LineItem.new
20   end
21
22   # GET /line_items/1/edit
23   def edit
24   end
25
26   # POST /line_items
```

```
27   # POST /line_items.json
28   def create
29     product = Product.find(params[:product_id])
30     @line_item = @cart.add_product(product)
31
32     respond_to do |format|
33       if @line_item.save
34         format.html { redirect_to @line_item.cart }
35         format.json { render :show,
36           status: :created, location: @line_item }
37       else
38         format.html { render :new }
39         format.json { render json: @line_item.errors,
40           status: :unprocessable_entity }
41       end
42     end
43   end
44
45   # PATCH/PUT /line_items/1
46   # PATCH/PUT /line_items/1.json
47   def update
48     respond_to do |format|
49       if @line_item.update(line_item_params)
50         format.html { redirect_to @line_item, notice: 'Line item was
               successfully updated.' }
51         format.json { render :show, status: :ok, location: @line_item }
52       else
53         format.html { render :edit }
54         format.json { render json: @line_item.errors, status: :
               unprocessable_entity }
55       end
56     end
57   end
58
59   # DELETE /line_items/1
60   # DELETE /line_items/1.json
61   def destroy
62     @line_item.destroy
63     respond_to do |format|
64       format.html { redirect_to line_items_url, notice: 'Line item was
               successfully destroyed.' }
65       format.json { head :no_content }
66     end
67   end
68
69   private
70     # Use callbacks to share common setup or constraints between actions
           .
71     def set_line_item
```

```
72        @line_item = LineItem.find(params[:id])
73      end
74
75      # Never trust parameters from the scary internet, only allow the
           white
76      # list through.
77      def line_item_params
78        params.require(:line_item).permit(:product_id)
79      end
80    #...
81  end
```

Ao invés de exibir os elementos de totalização do cart com <li> vamos utilizar uma tabela. Vamos utilizar CSS para criar um estilo. Assim, o arquivo qpp/views/-carts/show.html.erb ficará (veja as linhas de 3 a 18):

```
1  <p id="notice"><%= notice %></p>
2
3  <h2>Your Cart</h2>
4  <table>
5    <% @cart.line_items.each do |item| %>
6      <tr>
7        <td><%= item.quantity %>&times;</td>
8        <td><%= item.product.title %></td>
9        <td class="item_price"><%= number_to_currency(item.total_price) %>
            </td>
10     </tr>
11   <% end %>
12
13   <tr class="total_line">
14     <td colspan="2">Total</td>
15     <td class="total_cell"><%= number_to_currency(@cart.total_price) %><
          /td>
16   </tr>
17
18 </table>
19
20 <%= button_to 'Empty cart', @cart, method: :delete,
21     data: { confirm: 'Are you sure?' } %>
```

Para que ele funcione, devemos modificar o app/models/line_item.rb para que ele retorne o preço total do item (linha 9 - superior), (veja as linhas 5 a 7 - abaixo).

```
1  class LineItem < ApplicationRecord
2    belongs_to :product
3    belongs_to :cart
4
5    def total_price
```

```
6       product.price * quantity
7     end
8   end
```

Devemos modificar também o app/models/cart.rb para que ele retorne o preço total do cart (linha 15 - superior), (veja as linhas 14 a 16).

```
1   class Cart < ApplicationRecord
2     has_many :line_items, dependent: :destroy
3
4     def add_product(product)
5       current_item = line_items.find_by(product_id: product.id)
6       if current_item
7         current_item.quantity += 1
8       else
9         current_item = line_items.build(product_id: product.id)
10      end
11      current_item
12    end
13
14    def total_price
15      line_items.to_a.sum { |item| item.total_price }
16    end
17  end
```

Vamos acrescentar algum estilo em app/assets/stylesheets/cart.scss:

```
1   // Place all the styles related to the Carts controller here.
2   // They will automatically be included in application.css.
3   // You can use Sass (SCSS) here: http://sass-lang.com/
4   .carts {
5     .item_price {
6       text-align: right;
7     }
8
9     .total_line, .total_cell {
10      font-weight: bold;
11      border-top: 1px solid #595;
12    }
13  }
```

Vamos verificar commo ficou. A dificuldade é uma vez que o item foi inserido, como retornamos para a loja? Por enquanto vamos apenas mudar a URL.

Último detalhe, vamos alterar os testes em test/controllers/line_items_controller.rb para refletir as modificações efetuadas (veja as linhas 25 e 26).

```
1  require 'test_helper'
2
3  class LineItemsControllerTest < ActionDispatch::IntegrationTest
4    setup do
5      @line_item = line_items(:one)
6    end
7
8    test "should get index" do
9      get line_items_url
10     assert_response :success
11   end
12
13   test "should get new" do
14     get new_line_item_url
15     assert_response :success
16   end
17
18   test "should create line_item" do
19     assert_difference('LineItem.count') do
20       post line_items_url, params: { product_id: products(:ruby).id }
21     end
22
23     follow_redirect!
24
25     assert_select 'h2', 'Your Cart'
26     assert_select 'td', "Programming Ruby 1.9"
27   end
28
29   test "should show line_item" do
30     get line_item_url(@line_item)
31     assert_response :success
32   end
33
34   test "should get edit" do
35     get edit_line_item_url(@line_item)
36     assert_response :success
37   end
38
39   test "should update line_item" do
40     patch line_item_url(@line_item),
41       params: { line_item: { product_id: @line_item.product_id } }
42     assert_redirected_to line_item_url(@line_item)
43   end
44
45   test "should destroy line_item" do
46     assert_difference('LineItem.count', -1) do
47       delete line_item_url(@line_item)
48     end
49
```

```
50      assert_redirected_to line_items_url
51    end
52  end
```

Execute os testes agora, devem ser livre de erros.

```
1  rails test:controllers
```