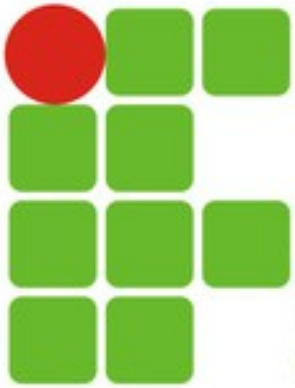

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

Destacando as Mudanças
(continuando a Tarefa F)

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

Feedback do Cliente

- Você me falou que adicionou AJAX... Mas, ao clicar nesse botão não vejo nada acontecer
- Mas, um novo item foi inserido... Veja aqui!
- Ok! Mas, mudou tão pouco, você não poderia destacar isso que mudou?

Iteração F3:

DESTACANDO AS MUDANÇAS

Javascript

- Várias bibliotecas Javascript estão incluídas no Rails – dentre elas a Jquery UI
  - Permite a decoração de páginas Web
  - Exemplo: técnica "*Yellow Fade*"

- Proposta: destacar o elemento recém inserido no carrinho com "fade"

Javascript

- Instalando a biblioteca JQuery UI

Download rails40/depot_m/Gemfile

```
# Use jquery as the JavaScript library
gem 'jquery-rails'
gem 'jquery-ui-rails'
```

  - Depois executa-se: ***bundle install***

- Basta incluir o efeito desejado... Para poder utilizar em um dado código JQuery

Javascript

Download rails40/depot_m/app/assets/javascripts/application.js

```
// This is a manifest file that'll be compiled into application.js, which will
// include all the files listed below.
//
// Any JavaScript/Coffee file within this directory, lib/assets/javascripts,
// vendor/assets/javascripts, or vendor/assets/javascripts of plugins, if any,
// can be referenced here using a relative path.
//
// It's not advisable to add code directly here, but if you do, it'll appear at
// the bottom of the compiled file.
//
// Read Sprockets README
// (https://github.com/sstephenson/sprockets#sprockets-directives) for details
// about supported directives.
//
//= require jquery
//= require jquery.ui.effect-blind
//= require jquery_ujs
//= require turbolinks
//= require_tree .
```

Identificando o Elemento a Destacar

- Cada linha de item do carrinho é representada em uma linha da tabela **<tr>**
  - É preciso marcar a linha do item recém inserido

- 1º passo – o controlador de linha de item informa o item que foi alterado – através da definição de uma variável de instância

Identificando o Elemento a Destacar

Download rails40/depot_m/app/controllers/line_items_controller.rb

```
def create
  product = Product.find(params[:product_id])
  @line_item = @cart.add_product(product.id)

  respond_to do |format|
    if @line_item.save
      format.html { redirect_to store_url }
      format.js   { @current_item = @line_item }
      format.json { render action: 'show',
        status: :created, location: @line_item }
    else
      format.html { render action: 'new' }
      format.json { render json: @line_item.errors,
        status: :unprocessable_entity }
    end
  end
end
```

Identificando o Elemento a Destacar

- Na “parcial” de linha de item é checado se esta corresponde ao “item corrente”
  - Caso afirmativo, é adicionado um **id** à **<tr>**

Download rails40/depot_m/app/views/line_items/_line_item.html.erb

```
<% if line_item == @current_item %>
<tr id="current_item">
<% else %>
<tr>
<% end %>
  <td><%= line_item.quantity %>&times;</td>
  <td><%= line_item.product.title %></td>
  <td class="item_price"><%= number_to_currency(line_item.total_price) %></td>
</tr>
```

Definindo o Efeito

- Com a marcação da linha desejada, no código JQuery que solicita a alteração do HTML do carrinho, também é definido o efeito desejado

Download rails40/depot_m/app/views/line_items/create.js.erb

```
$('#cart').html("<%= escape_javascript render(@cart) %>");

$('#current_item').css({'background-color':'#88ff88'}).
  animate({'background-color':'#114411'}, 1000);
```

Iteração F4:

ESCONDENDO UM CARRINHO VAZIO

Última Requisição do Cliente

- Será que não podemos esconder o carrinho de compras quando este não tiver itens?
  - Claro que sim!
  - Considere feito!

Escondendo o Carrinho

- Pode ser realizado de uma série de formas

- Mais simples: só incluir o HTML do carrinho quando este não estiver vazio
  - Podemos resolver isso na "parcial" do carrinho

Escondendo o Carrinho

```
<% unless cart.line_items.empty? %>
<div class="cart_title">Your Cart</div>
<table>
  <%= render(cart.line_items) %>

  <tr class="total_line">
    <td colspan="2">Total</td>
    <td class="total_cell"><%= number_to_currency(cart.total_price) %></td>
  </tr>
</table>

<%= button_to 'Empty cart', cart, method: :delete,
    confirm: 'Are you sure?' %>
<% end %>
```

Escondendo o Carrinho

- Porém essa estratégia irá causar o carregamento do carrinho inteiro ao adicionarmos o primeiro item

- Vamos buscar outra estratégia...

Escondendo o Carrinho

- JQuery UI também oferece “transições” que definem como um elemento “entra em cena”
  - Nesse caso, desejamos “revelar” o carrinho de compras quando o mesmo tiver um item

Download rails40/depot_n/app/views/line_items/create.js.erb

```
if ($('#cart tr').length == 1) { $('#cart').show('blind', 1000); }

$('#cart').html("<%= escape_javascript render(@cart) %>");

$('#current_item').css({'background-color':'#88ff88'}).
  animate({'background-color':'#114411'}, 1000);
```

Escondendo o Carrinho

- Também precisamos nos preocupar em esconder o carrinho quando este estiver vazio

- Dada a experiência da iteração, temos que a melhor prática seria
  - Criar o HTML para o carrinho
  - Definir ***display: none*** se o carrinho estiver vazio

Escondendo o Carrinho

```
<div id="cart"
    <% if @cart.line_items.empty? %>
        style="display: none"
    <% end %>
  >
  <%= render(@cart) %>
</div>
```

- Porém essa abordagem não gera um código muito legível – aparenta estar faltando alguma coisa
- Solução: criar um método auxiliar (*helper method*) para abstrair esse problema

Helper Method

- São úteis quando se deseja abstrair algum procedimento para fora de uma visão
- O diretório da aplicação possui seis subdiretórios

```
depot> ls -p app
assets/  controllers/  helpers/  mailers/  models/  views/

depot> ls -p app/helpers
application_helper.rb  line_items_helper.rb  store_helper.rb
carts_helper.rb        products_helper.rb
```

Helper Method

- O gerador de código Rails criou um arquivo *helper* para cada um dos controladores
- Como será utilizado no layout da aplicação, o mesmo irá para o *helper* de aplicação
- Será chamado da seguinte forma:

Download rails40/depot_n/app/views/layouts/application.html.erb

```
<%= hidden_div_if(@cart.line_items.empty?, id: 'cart') do %>
  <%= render @cart %>
<% end %>
```

Helper Method

- Vejamos o código do mesmo:

Download rails40/depot_n/app/helpers/application_helper.rb

```
module ApplicationHelper
  def hidden_div_if(condition, attributes = {}, &block)
    if condition
      attributes["style"] = "display: none"
    end
    content_tag("div", attributes, &block)
  end
end
```

Helper Method

- Esse código utiliza um método auxiliar padrão do Rails – **content_tag()**
  - Encapsula o resultado do método em uma *tag*
  - O bloco a ser manipulado foi passado como argumento para o método

Últimos Ajustes

- Finalmente, não precisamos mais exibir a mensagem quando o carrinho é esvaziado
  - Primeiro, porque o carrinho irá simplesmente desaparecer quando for esvaziado
  - Segundo, agora que está sendo utilizado AJAX, a página do catálogo não é recarregada
    - Uma vez que a mensagem for exibida, ela pode ficar lá mesmo quando itens forem inseridos no carrinho

Últimos Ajustes

- Deixando de exibir a notificação de que o "carrinho foi esvaziado"

Download rails40/depot_n/app/controllers/carts_controller.rb

```
def destroy
  @cart.destroy if @cart.id == session[:cart_id]
  session[:cart_id] = nil
  respond_to do |format|
    format.html { redirect_to store_url }
    format.json { head :no_content }
  end
end
```

Iteração F5:

FAZENDO AS IMAGENS SEREM “CLICÁVEIS”

Clicar em uma Imagem

- Queremos que uma imagem manipule o evento de "**onClick**"
  - Queremos preparar um script que execute quando a página carregar que encontra todas as imagens

- Como está organizada tal página?

Organização do Catálogo

Download rails40/depot_n/app/views/store/index.html.erb

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

<h1>Your Pragmatic Catalog</h1>

<% cache ['store', Product.latest] do %>
  <% @products.each do |product| %>
    <% cache ['entry', 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),
          remote: true %>
      </div>
    </div>
  <% end %>
<% end %>
<% end %>
```

Tornando as Imagens Clicáveis

- Modificação realizada por meio de Javascript

Download rails40/depot_n/app/assets/javascripts/store.js.coffee

```
# Place all the behaviors and hooks related to the matching controller here.
# All this logic will automatically be available in application.js.
# You can use CoffeeScript in this file: http://coffeescript.org/

$(document).on "ready page:change", ->
  $('.store .entry > img').click ->
    $(this).parent().find(':submit').click()
```

  - **CoffeScript** é mais um pré-processador Javascript
  - Combinado com JQuery produz efeitos interessantes com poucas linhas de código

Tornando as Imagens Clicáveis

- O que foi realizado:
  - Função a ser executada quando a página carregar
  - Associada a dois eventos “*ready*” e “*page:change*”
  - Encontra elementos da classe “*entry*” (que descendem da classe “*store*”)
  - Para cada imagem encontrada, associa eventos de clique as mesmas
  - Define que o clique na imagem será tratado pela “entrada” correspondente

Resultado

- O layout não mudou, mas...
- Ao clicar na imagem de um produto, o mesmo é inserido no carrinho de compras

- Funcionalidade implementada → testar!
  - Ops! O teste indica uma série de erros
  - O uso do AJAX implicará implicará em adequações