



Ruby on Rails 3.2.2: Criando uma loja

Brayan Neves



Universidade Federal
de Ouro Preto



Criando e configurando um novo projeto

- ▶ Modelagem
- ▶ Criação do projeto
- ▶ Configuração
- ▶ Criação de um novo componente
- ▶ Visualização do novo componente
- ▶ Editando campos de inserção
- ▶ Aplicando arquivos públicos

Criação de um novo projeto

- ▶ Para criar um novo projeto usar os comandos:
 - rails new Shop -d=mysql
 - -d (database): vamos usar o mysql como banco de dados do projeto

Configuração

- ▶ Precisamos configurar nosso aplicativo para ele se comunicar com o banco de dados
 - Editar: config/database.yml
 - Definir senha em **development** e **test**
 - A senha de nossa VM é: **rootadmin**
 - Criando banco de dados
 - Executar o comando: `rake db:create`

Criando um novo componente

- ▶ Para criar um novo componente, vamos usar o comando scaffold
 - rails generate scaffold Product \
title:string description:text \
image_url:string price:decimal

Criando um novo componente

- ▶ Agora precisamos publicar nosso componente na base de dados, mas vamos editar os parâmetros do preço
 - Editar db/migrate/2012..._create_products.rb
 - t.decimal :price, :precision =>8, scale => 2
 - Executar: rake db:migrate
 - Este comando deve ser executado toda vez que se cria uma nova entidade

Visualização do novo componente

- ▶ Para ver o que nosso programa gerou temos que iniciar o nosso servidor
 - rails server -d
 - -d (daemon): executa o servidor em segundo plano
 - Acessar a página do componente
 - <http://localhost:3000/products/>

Editando campos de inserção

- ▶ Para melhor visualização dos dados que devemos inserir vamos reduzir o tamanho do campo de descrição
 - Alterar arquivo: app/views/products/_form.html.erb
 - ```
<div class="field">
 <%= f.label :title %>
 <%= f.text_area :description, :rows => 6 %>
</div>
```

# Aplicação de arquivos publicos

- ▶ CSS
  - Carregar shop.css para app/assets/stylesheets
- ▶ Imagens
  - Carregar imagens para app/assets/images
- ▶ Modificando lista de produtos
  - Alterar index.html.erb aplicando os estilos

# Validação

- ▶ Para termos uma base de dados consistente, precisamos criar algumas regras de negócio.
  - Alterar: app/models/products\_controller.rb
  - validates :title, :description, :image\_url, :presence => true
    - Validar se título, descrição e imagem estão presentes
  - validates :price, :numericality => { :greater\_than\_or\_equal\_to => 0.00}
    - Validar se o preço é no mínimo R\$ 0.00

# Validação

- ▶ Para termos uma base de dados consistente, precisamos criar algumas regras de negócio.
  - Alterar: app/models/products\_controller.rb
  - validates :title, :uniqueness => true
    - Validar se título é único
  - validates :image\_url, :format => {  
    :with => %r{\.(gif|jpg|png)\$}I,  
    :message => 'imagem precisa ser um GIF, PNG ou JPG'  
}
  - Validar imagem para GIF, PNG ou JPG

# Teste de Validação

- ▶ Antes de continuar vamos testar nossas validações
  - rake test
- ▶ Deveria ser encontradas duas falhas
  - test\_should\_create\_product
  - test\_should\_update\_product
- Dados de teste não são válidos

# Teste de Validação

- ▶ Temos que inserir dados válidos no arquivo de teste

- Editar: test/functional/products\_controller\_test.rb
  - Construtor

- setup do

```
@product = products(:one)
@update = {
 :title => 'Gado' ,
 :description => 'Gado de corte' ,
 :image_url => 'gado.jpg' ,
 :price => 2650.00
}
end
```

# Teste de Validação

- ▶ Temos que inserir dados válidos no arquivo de teste
  - Editar: test/functional/products\_controller\_test.rb
  - should\_create\_product
    - test "should create product" do
    - assert\_difference('Product.count') do
    - post :create, **product: @update**
    - end
  - should\_update\_product
    - test "should update product" do
    - put :update, id: **@product.to\_param**, product: **@update**
    - assert\_redirected\_to product\_path(assigns(:product))
    - end

# Criando um catálogo

- ▶ Já criamos um controle de produtos para o vendedor administrar seus produtos, agora vamos criar um catálogo para os compradores, chamaremos de store
  - Executar: rails generate store index
- ▶ Foi criado um controle `store.rb` em nossa pasta de controllers com uma única função (`index`) que pode ser acessada no comando  
<http://localhost:3000/store/index>

# Criando um catálogo

- ▶ Agora vamos setar index como a nova página principal de nossa aplicação
  - Em: config/routes.rb, adicionar as linhas:
    - root :to => 'store#index', :as => 'store'
  - Remover antiga página inicial
    - rm public/index.html

# Criando um catálogo

- ▶ Agora vamos modificar a função index para exibir nossos produtos
  - Editar: app/controllers/store\_controller.rb
    - def index

```
@products = Product.all
end
```
    - Para criar uma lista
  - Inserir em: app/models/store.rb
    - default\_scope :order => 'title'
    - Para definir a ordem da lista (alfabética)

# Criando um catálogo

- ▶ Agora vamos modificar a função index para exibir nossos produtos

- Editar: app/views/store/index.html.erb

- ```
<% if notice %>
  <p id="notice" ><%= notice %></p>
<% end %>
<h1>Your Pragmatic Catalog</h1>
<% @products.each do |product| %>
  <div class="entry" >
    <%= image_tag(product.image_url) %>
    <h3><%= product.title %></h3>
    <%= sanitize(product.description) %>
    <div class="price_line" >
      <span class="price" ><%= product.price %></span>
    </div>
  </div>
<% end %>
```

Criando um catálogo

- ▶ Modificar a página principal de nosso aplicativo
 - Editar o body de: app/views/layouts/application.html.erb
 - <body id="store" >
 <div id="banner" >
 <%= image_tag("logo.png") %>
 <%= @page_title || "Pragmatic Bookshelf" %>
 </div>
 <div id="columns" >
 <div id="side" >
 Home

 Questions

 News

 Contact

 </div>
 <div id="main" >
 <%= yield %>
 </div>
 </div>
 </body>

Criando um catálogo

- ▶ Aplicar estilo (CSS) na página
 - Substituir o arquivo shop.css na pasta de estilos

Criando um Carrinho

- ▶ Para criar o carrinho de compras vamos usar o comando scaffold novamente.
 - rails generate scaffold cart
 - rake db:migrate
- ▶ O carrinho representará uma seção aberta da loja, modificaremos o controller do aplicativo para fazer isso
 - Editar: app/controller/application_controller.erb

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

Criando um Carrinho

- ▶ Para conectar um produto em um carrinho criaremos um link
 - rails generate scaffold line_item \
product_id:integer cart_id:integer
 - rake db:migrate

Criando um carrinho

- ▶ Agora precisamos fazer as associações nos modelos

- Editar: app/models/cart.rb

- class Cart < ActiveRecord::Base
 has_many :line_items, :dependent => :destroy
 end

- Editar: app/models/line_item.rb

- class LineItem < ActiveRecord::Base
 belongs_to :product
 belongs_to :cart
 end

Criando um carrinho

- ▶ Com esses ligações, agora temos objetos associados, por exemplo:
 - Podemos executar comandos como:
 - `line_item.product.title`
 - `cart.line_items.count`

Criando um carrinho

► Finalmente linkar um produto a um line_item

- has_many :line_items
- before_destroy :**ensure_not_referenced_by_any_line_item**
- private
 - # ensure that there are no line items referencing this product
 - def **ensure_not_referenced_by_any_line_item**
 - if line_items.empty?
 - return true
 - else
 - errors.add(:base, 'Line Items present')
 - return false
 - end

Criando um carrinho

- ▶ Vamos agora adicionar o botão “Comprar” para o usuário poder comprar itens
 - Editar app/views/store/index.html.erb
 - ```
<div class="price_line">
 <%= number_to_currency(product.price) %>
 <%= button_to 'Add to Cart',
 line_items_path(:product_id => product) %>
</div>
```
  - Esse botão irá gerar um comando POST para line\_item enviando um id de produto como parametro

# Criando um carrinho

- ▶ Para colocar o botão de comprar na frente de nosso preço vamos adicionar uma regras em nosso shop.css

- ```
#store .entry form, #store .entry form div {  
    display: inline;  
}
```

Criando um carrinho

- ▶ Agora precisamos definir nossa regra de criação de links para que eles sejam registrados no banco.
 - Editar: app/controllers/line_item_controller.rb
 - ```
def create
 @cart = current_cart
 product = Product.find(params[:product_id])
 @line_item = @cart.line_items.build(:product => product)
 respond_to do |format|
 if @line_item.save
 format.html { redirect_to(@line_item.cart,
 :notice => 'Line item was successfully created.') }
 format.xml { ... }
 else
 ...
 end
 end
end
```

# Criando um carrinho

- ▶ E finalmente criaremos a visualização de nosso carrinho.

- Editar: app/views/carts/show.html.erb
    - <h2>Your Pragmatic Cart</h2>
    - <ul>
    - <% @cart.line\_items.each do |item| %>
    - <li><%= item.product.title %></li>
    - <% end %>
    - </ul>

# Deixando o carrinho inteligente

- ▶ Adicionando quantidade de itens ao carrinho
  - Executar: `rails generate migration add_quantity_to_line_items quantity:integer`
  - Editar: `db/migrate/2012...add_quantity_to_line_items`
    - `def self.up`  
    `add_column :line_items, :quantity, :integer, :default => 1`  
`end`
  - Para completar, executar: `rake db:migrate`

# Deixando o carrinho inteligente

- Deixando o botão de comprar mais inteligente:

- Adicionar em: app/models/cart.rb

```
• def add_product(product_id)
 current_item =
line_items.find_by_product_id(product_id)
 if current_item
 current_item.quantity += 1
 else
 current_item = line_items.build(:product_id =>
product_id)
 end
 current_item
end
```

# Deixando o carrinho inteligente

- ▶ Com a função de adição criada, vamos mudar a linha onde criamos o nosso line\_item
  - Editar: app/controllers/line\_items\_controller.rb
    - def create@cart = current\_cartproduct =Product.find(params[:product\_id])**@line\_item = @cart.add\_product(product.id)**respond\_to do |format|  
...  
...

# Deixando o carrinho inteligente

- ▶ E vamos fazer uma pequena mudança no show de nosso carrinho
  - Editar: app/views/carts/show.html.erb
    - <h2>Your Pragmatic Cart</h2><ul><% @cart.line\_items.each do |item| %><li><%= item.quantity %> &times; <%= item.product.title %></li><% end %></ul>

# Corrigindo o banco de dados

## ▶ Para corrigir o banco, faremos uma migração

- Executar: rails generate migration combine\_items\_in\_cart
- E vamos editá-la: db/migrate/2012...combine\_items\_in\_cart

```
• def self.up
 Cart.all.each do |cart|
 sums = cart.line_items.group(:product_id).sum(:quantity)
 sums.each do |product_id, quantity|
 if quantity > 1
 cart.line_items.where(:product_id=>product_id).delete_all
 end
 end
 end
end
```

- Por fim: rake db:migrate

# Corrigindo o banco de dados

- ▶ Toda operação em rails deve ser reversível, então mudaremos a função de volta

- ```
def self.down
    LineItem.where("quantity>1").each do |line_item|
        line_item.quantity.times do
            LineItem.create :cart_id=>line_item.cart_id,
                            :product_id=>line_item.product_id, :quantity=>1
        end
        line_item.destroy
    end
end
```

- Para reverter, utilizamos: `rake db:rollback`

Tratando erro de id

- ▶ **Editar:** app/controllers/carts_controller.rb

- def show
 - begin**
 - @cart = Cart.find(params[:id])
 - rescue ActiveRecord::RecordNotFound**
 - logger.error "Attempt to access invalid cart
#{params[:id]}"
 - redirect_to store_url, :notice => 'Invalid cart'**
 - else**
 - respond_to do |format|
 - format.html # show.html.erb
 - format.xml { render :xml => @cart }
 - end
 - end**
 - end

Finalizando o Carrinho

► Botão de limpar carrinho

- Inserir em: app/views/carts/show.html.erb
 - `<%= button_to 'Empty cart', @cart, :method => :delete, :confirm => 'Are you sure?' %>`
- Garantir que quando um carrinho for limpo, “finalize a seção” e redirecione para a loja, editar: app/controllers/carts_controller.rb
 - `def destroy
 @cart = current_cart
 @cart.destroy
 session[:cart_id] = nil
 respond_to do |format|
 format.html { redirect_to(store_url, :notice => 'Your cart is currently empty') }
 format.xml { head :ok }
 end
end`

Finalizando o Carrinho

► Aplicando CSS no carrinho – show do cart

- ```
<div class="cart_title" >Your Cart</div>
<table>
 <% @cart.line_items.each do |item| %>
 <tr>
 <td><%= item.quantity %>×</td>
 <td><%= item.product.title %></td>
 <td class="item_price" ><%
 number_to_currency(item.total_price) %></td>
 </tr>
 <% end %>
 <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?' %>
```

# Finalizando o Carrinho

## ► Definindo função de soma

- app/models/line\_item.rb
  - def total\_price
  - product.price \* quantity
  - end
- app/models/cart.rb
  - def total\_price
  - line\_items.to\_a.sum { |item| item.total\_price }
  - end

# Finalizando o Carrinho

## ► Editando o CSS: app/assets/stylesheets/shop.css

```
◦ #store .cart_title {
 font: 120% bold;
}
#store .item_price, #store .total_line {
 text-align: right;
}
#store .total_line .total_cell {
 font-weight: bold;
 border-top: 1px solid #559;
}
```