

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

# Criando o Carrinho de Compras (Tarefa D)

Prof. Fellipe Aleixo ([fellipe.aleixo@ifrn.edu.br](mailto:fellipe.aleixo@ifrn.edu.br))

# Feedback do Cliente

- “Legal, temos o catálogo de produtos! Seria muito legal que pudéssemos estar preparados para vender os mesmos”
  - “Que tal começarmos pelo carrinho de compras?”
- Ok!
  - Essa funcionalidade irá implicar em vários novos conceitos: sessões, relacionamento entre modelos, adicionando botões à visão e etc.

Iteração D1:

# **ENCONTRANDO UM CARRINHO**

# Carrinho de Compras

- Os consumidores irão navegar no catálogo na esperança de encontrarem o que desejam
  - Ao encontrar tais produtos, deverão guardar os mesmos em um carrinho de compras
  - Ao encontrarem o que desejam, estes irão realizar o *checkout*, e pagar pelos itens no carrinho
- O carrinho será armazenado no banco, e será guardado um identificador do mesmo na sessão

# Criando o Carrinho de Compras

```
depot> rails generate scaffold Cart
...
depot> rake db:migrate
== CreateCarts: migrating =====
-- create_table(:carts)
   -> 0.0012s
== CreateCarts: migrated (0.0014s) =====
```

# Criando o Carrinho de Compras

- Em Rails, a sessão atual tem a forma de *hash*
  - O identificador do carrinho de compras será armazenado na sessão, e indexado pelo símbolo **:cart\_id**

# Armazenando o Identificador na Sessão

Download rails40/depot\_f/app/controllers/concerns/current\_cart.rb

```
module CurrentCart
  extend ActiveSupport::Concern

  private

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

# Trabalhando com Módulos

- O método **set\_cart()** foi definido em um módulo e marcado como privado
  - Essa abordagem permite o compartilhamento de código entre controladores, mesmo que seja um único método



Iteração D2:

# **CONECTANDO PRODUTOS AO CARRINHO**

# Relacionando Produtos ao Carrinho

```
depot> rails generate scaffold LineItem product:references cart:belongs_to
...
depot> rake db:migrate
== CreateLineItems: migrating =====
-- create_table(:line_items)
   -> 0.0013s
== CreateLineItems: migrated (0.0014s) =====
```

- Elemento de modelo: “Linha de Item”
- Referencia um produto, e
- Faz parte de um carrinho

# Relacionando Produtos ao Carrinho

- A definição dos relacionamentos na classe de modelo LineItem

Download rails40/depot\_f/app/models/line\_item.rb

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

# Especificando Relacionamentos

- As chamadas do **belong\_to()**, informam ao Rails que linhas da tabela **line\_item** são relacionadas às tabelas **cart** e **product**
- Nenhuma linha de item pode existir sem que existam o carrinho e o produto referenciados

# Navegando nos Relacionamentos

- A definição de relacionamentos possibilita a navegação pelos elementos de modelo
- Dado o relacionamento entre **line\_item** e **product**, é possível acessar o produto associado à linha de item

# Navegando nos Relacionamentos

- Exemplo:

```
li = LineItem.find(...)
puts "This line item is for #{li.product.title}"
```

- Em caso de relacionamentos bidirecionais:

[Download rails40/depot\\_f/app/models/cart.rb](#)

```
class Cart < ActiveRecord::Base
  ➤ has_many :line_items, dependent: :destroy
end
```

# Relacionamentos para Coleções

- Depois de declarado o relacionamento:

```
cart = Cart.find(...)  
puts "This cart has #{cart.line_items.count} line items"
```

## Download rails40/depot\_f/app/models/product.rb

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

```
➤ 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
```

```
➤   end
```

```
end
```



# Método *Hook*

- Um método *hook* é chamado pelo Rails automaticamente
  - No caso do exemplo: chamado antes que Rails delete a linha correspondente do banco de dados
    - Se retornar “falso” a linha não será deletada
  - Note que há um acesso direto ao objeto **errors**
    - Mesmo lugar que o **validates()** armazena os erros

Iteração D3:

# **ADICIONANDO UM BOTÃO**

# Próximo Passo

- Ok, carrinho criado, é hora de adicionar a capacidade de adicionar itens no mesmo
  - Criar o botão “Adicionar ao carrinho” para cada um dos itens do catálogo
  - Há a necessidade de criar um novo controlador?

# Ações do Controlador do Carrinho de Compras

- Ações criadas para o controlador das “linhas de item” na geração do esqueleto de CRUD
  - app/controllers/line\_items\_controller.rb
  - index(), show(), new(), edit(), create() e update()
- A adição de um novo item ao carrinho equivale a operação **create()**
  - A ação **new()** é responsável pelo formulário que aciona o **create()**

# Adicionando Botões

- A inclusão de um botão em um ERB é realizada através do método **button\_to()**
  - O botão pode ser conectado à linha de item através da especificação da URL
    - **line\_items\_path**
  - Como saber que produto deve ser adicionado no carrinho de compras?
    - Passando o ID do produto
  - Vejamos o código...

## Download rails40/depot\_f/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) %>
        </div>
      </div>
    <% end %>
  <% end %>
<% end %>
```

# Adicionando Botões

- Falta apenas um ajuste na formatação
  - O método **button\_to()** gera um formulário (HTML)
  - O formulário gerado contém um `<div>`
  - Por isso – o botão é exibido em uma nova linha

```
Download rails40/depot_f/app/assets/stylesheets/store.css.scss
```

```
p, div.price_line {  
  margin-left: 100px;  
  margin-top: 0.5em;  
  margin-bottom: 0.8em;
```

```
> form, div {  
>   display: inline;  
> }  
}
```

# Resultado

Pragprog Books Online Store

localhost:3000


**Pragmatic Bookshelf**

## PRAGMATIC BOOKSHELF

[Home](#)  
[Questions](#)  
[News](#)  
[Contact](#)

### Your Pragmatic Catalog

---

 **CoffeeScript**

CoffeeScript is JavaScript done right. It provides all of JavaScript's functionality wrapped in a cleaner, more succinct syntax. In the first book on this exciting new language, CoffeeScript guru Trevor Burnham shows you how to hold onto all the power and flexibility of JavaScript while writing clearer, cleaner, and safer code.

**\$36.00**



# Modificando o Método `create()`

Download rails40/depot\_f/app/controllers/line\_items\_controller.rb

```
class LineItemsController < ApplicationController
```

```
➤ include CurrentCart
```

```
➤ before_action :set_cart, only: [:create]
```

```
before_action :set_line_item, only: [:show, :edit, :update, :destroy]
```

```
# GET /line_items
```

```
#...
```

```
end
```

- Antes de adicionar um item, é necessário recuperar o carrinho de compras da sessão
- Definir **`set_cart()`** para executar antes de **`create()`**

# Modificando o Método create()

Download rails40/depot\_f/app/controllers/line\_items\_controller.rb

```
def create
```

- 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.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
```

# Modificando o Método `create()`

- Utilizado o objeto `params` para recuperar o `:product_id`
  - Inclui os parâmetros enviados na requisição
- `@cart.line_items.build()` → cria um novo relacionamento (carrinho ↔ produto)
  - Resultado armazenado em `@line_item`
- Após a inserção com sucesso a resposta será redirecionada para o carrinho correspondente

# Inserir o Teste da Nova Funcionalidade

- Após da modificação do controlador, é hora de incluir os testes para tal funcionalidade
- Qual é mesmo o tipo de teste necessário?
  - Teste funcional
- O teste deverá simular a passagem de um id de produto para o método **create()**
  - Verificar se ocorre o redirecionamento (sucesso)

# Inserir o Teste da Nova Funcionalidade

```
Download rails40/depot_g/test/controllers/line_items_controller_test.rb
```

```
test "should create line_item" do
```

```
  assert_difference('LineItem.count') do
```

```
➤    post :create, product_id: products(:ruby).id
```

```
  end
```

```
➤    assert_redirected_to cart_path(assigns(:line_item).cart)
```

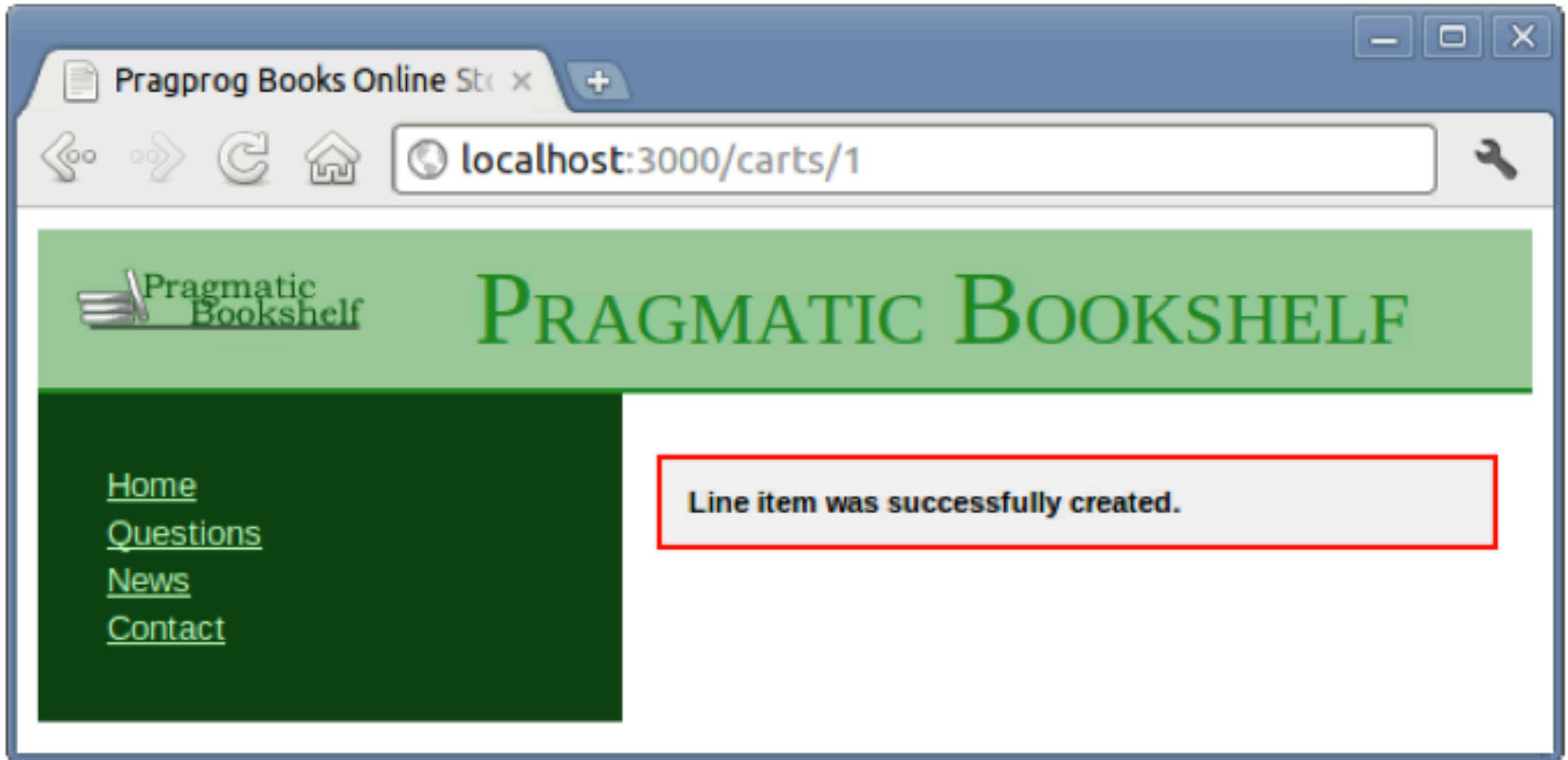
```
  end
```

– Simular método POST → **post**

– Verifica o redirecionamento → **assert\_redirect\_to**

```
depot> rake test test/controllers/line_items_controller_test.rb
```

# Resultado



# Exibindo os Dados do Carrinho

- Só foi exibida a mensagem porque a visualização do carrinho não foi alterada

Download rails40/depot\_f/app/views/carts/show.html.erb

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

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

# Exibindo os Dados do Carrinho

