

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

# Exibindo o Catálogo (Tarefa C)

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

Iteração C1:

# **CRIANDO A LISTAGEM DO CATÁLOGO**

# Feedback do Cliente

- O cliente foi questionado sobre “prioridades”
  - “Gostaria de ver como a aplicação irá funcionar do ponto de vista do comprador”
- Tarefa: criar a exibição do catálogo de produtos (tela inicial do comprador)

# Criando um Novo Controlador

- Já foi criado um controlador para Produto
  - Utilizado pelo vendedor para administrar a aplicação (manter o cadastro de produtos)
- É necessário um novo controlador
  - Interação com os consumidores
  - Será chamado de **Store**

# Criando um Novo Controlador

```
depot> rails generate controller Store index
create  app/controllers/store_controller.rb
route  get "store/index"
invoke  erb
create  app/views/store
create  app/views/store/index.html.erb
invoke  test_unit
create  test/controllers/store_controller_test.rb
invoke  helper
create  app/helpers/store_helper.rb
invoke  test_unit
create  test/helpers/store_helper_test.rb
invoke  assets
invoke  coffee
create  app/assets/javascripts/store.js.coffee
invoke  scss
create  app/assets/stylesheets/store.css.scss
```

# Criando um Novo Controlador

- Foi criada a classe StoreController
  - store\_controler.rb
  - Com um único método de ação: **index()**
  - Acessada por: **http:// localhost:3000/store/index**
  - Vamos simplificar as coisas para os usuários – tornando esta a URL raiz para a aplicação
    - Editar o arquivo **config/routes.rb**

# Criando um Novo Controlador

Download rails40/depot\_d/config/routes.rb

```
Depot::Application.routes.draw do
```

```
  get "store/index"
```

```
  resources :products
```

```
  # The priority is based upon order of creation:
```

```
  # first created -> highest priority.
```

```
  # See how all your routes lay out with "rake routes".
```

```
  # You can have the root of your site routed with "root"
```

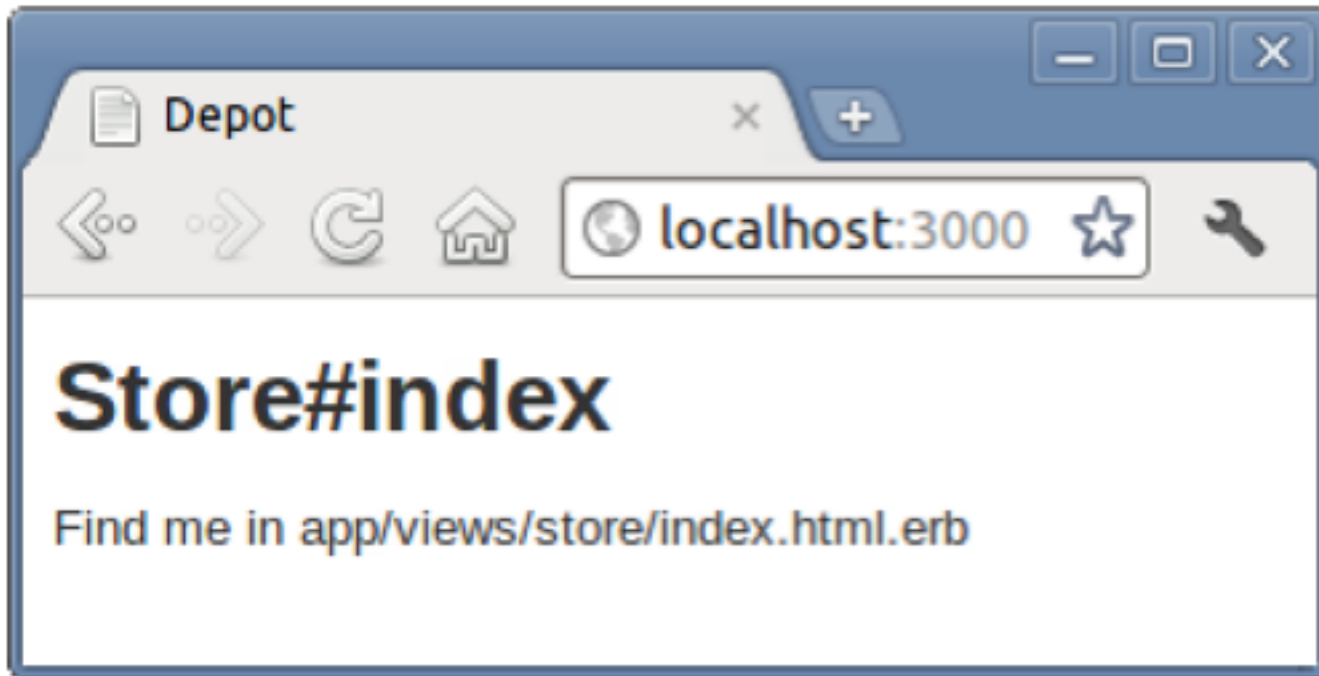
```
➤ root 'store#index', as: 'store'
```

```
  # ...
```

```
end
```

# Criando um Novo Controlador

- Agora, ao acessar `http://localhost:3000`





# Adicionando Funcionalidade

- Vamos iniciar exibindo uma simples lista dos produtos armazenados no banco de dados
  - Precisamos alterar o método **index()**

Download rails40/depot\_d/app/controllers/store\_controller.rb

```
class StoreController < ApplicationController
  def index
    ➤ @products = Product.order(:title)
  end
end
```

- decidimos juntos ver como os mesmos serão exibidos se usarmos a ordem alfabética

# Adicionando Funcionalidade

- Necessário escrever um *template* de visão
  - Editar o arquivo **index.html.erb** do diretório **app/views/store**
  - Os ajustes na aparência deve ser feitos no arquivo **app/assets/stylesheets/store.css.scss**

# Adicionando Funcionalidade

Download rails40/depot\_d/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 %>
```

# Adicionando Funcionalidade

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

```
// 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;
➤   }
➤
➤   h3 {
➤     font-size: 120%;
➤     font-family: sans-serif;
➤     margin-left: 100px;
➤     margin-top: 0;
➤     margin-bottom: 2px;
➤     color: #227;
➤   }
➤
➤   p, div.price_line {
➤     margin-left: 100px;
➤     margin-top: 0.5em;
➤     margin-bottom: 0.8em;
➤   }
➤
➤   .price {
➤     color: #44a;
➤     font-weight: bold;
➤     margin-right: 3em;
➤   }
➤ }
➤ }
```

# Resultado



# Feedback do Cliente

- “Ainda está muito básico, e parece estar faltando algo!”
- “Seria interessante criar meios de auxiliar na navegação do consumidor – ajudá-lo a achar aquilo que ele deseja”

Iteração C2:

# **ADICIONANDO UM LAYOUT DE PÁGINA**



# Layout da Aplicação

- As páginas de um site Web geralmente compartilham um layout similar
  - *Template* padrão criado por um *designer*
  - O trabalho do desenvolvedor é então – modificar as páginas e adicionar a decoração as mesmas

# Adicionando Elementos em Todas as Páginas da Aplicação

- Elementos a serem adicionados
  - Banner
  - Menu lateral
- Necessário editar o arquivo **`app/views/layouts/application.html.erb`**

```
Line 1 <!DOCTYPE html>
- <html>
- <head>
-   <title>Pragprog Books Online Store</title>
5   <%= stylesheet_link_tag "application", media: "all",
-     "data-turbolinks-track" => true %>
-   <%= javascript_include_tag "application", "data-turbolinks-track" => true %>
-   <%= csrf_meta_tags %>
- </head>
10 <body class="<%= controller.controller_name %>">
-   <div id="banner">
-     <%= image_tag("logo.png") %>
-     <%= @page_title || "Pragmatic Bookshelf" %>
-   </div>
15 <div id="columns">
-   <div id="side">
-     <ul>
-       <li><a href="http://www....">Home</a></li>
-       <li><a href="http://www..../faq">Questions</a></li>
20       <li><a href="http://www..../news">News</a></li>
-       <li><a href="http://www..../contact">Contact</a></li>
-     </ul>
-   </div>
-   <div id="main">
25     <%= yield %>
-   </div>
- </div>
- </body>
- </html>
```

# Adicionando Elementos em Todas as Páginas da Aplicação

- Itens específicos do Rails:
  - `stylesheet_link_tag()` → gera um tag `<link>`
  - `javascript_include_tag()` → gera um tag `<script>`
  - Uso da variável de instância `@page_title`
  - `yield()` → substitui o conteúdo por **`index.html.erb`**
- Os ajustes na aparência são realizados em **`app/assets/stylesheets/application.css.scss`**

# Adicionando Elementos em Todas as Páginas da Aplicação

- O layout criado consiste em três áreas
  - Um banner no topo
  - A área principal no centro a esquerda
  - Menu no centro a direita
- Todos os elementos devidamente decorados com referência à classes do CSS
  - O banner e o menu lateral

# 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.

**36.0**

# Feedback do Cliente

- “o valor dos produtos poderia ser exibido na notação correta”
  - “\$ 12.34” ao invés de “12.34”
  - Em Real teríamos “R\$ 12,34”

Iteração C3:

# **USANDO CLASSE AUXILIAR (HELPER) PARA FORMATAR O PREÇO**



# Formatação do Preço

- Pode ser utilizada a função **sprintf()** de Ruby
- Pode ser colocada a “lógica” na view

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

- E se precisarmos exibir preços em vários lugares?
  - E quando for para internacionalizar a aplicação?
- 
- Melhor opção – utilizar um método **helper** para formatar o preço com base na moeda

# Formatação do Preço

- Utilizando uma função *helper*, trocamos:

```
<span class="price"><%= product.price %></span>
```

– por


Download rails40/depot\_e/app/views/store/index.html.erb

```
<span class="price"><%= number_to_currency(product.price) %></span>
```

# Resultado

Pragprog Books Online Str x


localhost:3000

 **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**

Iteração C4:

# **TESTE FUNCIONAL DOS CONTROLADORES**

# Mais Testes

- Só podemos dizer que uma iteração foi finalizada com sucesso se passarem os testes
  - Antes de incluir novos testes, é bom rodar os testes e conferir os resultados
    - Verificar se até o momento não quebramos nada
- ```
depot> rake test
```
- Devemos adicionar testes para as partes que acabamos de inserir

# Testando os Demais Elementos

- Teste unitário de elementos de modelo: chamamos cada método e checamos se é retornado um resultado esperado
- Como criar testes unitários para os demais elementos – controladores e visões?
- O que precisamos é de testes funcionais
  - Se os elementos (MVC) funcionam bem juntos

# Testes Funcionais

- Vejamos o que Rails já gerou:

```
Download rails40/depot_d/test/controllers/store_controller_test.rb
```

```
require 'test_helper'
```

```
class StoreControllerTest < ActionController::TestCase
```

```
  test "should get index" do
```

```
    get :index
```

```
    assert_response :success
```

```
  end
```

```
end
```

# Testes Funcionais

- Testamos o resultado de uma requisição e verificamos se é a resposta esperada
- Possíveis verificações
  - Layout correto
  - Informações corretas do produto
  - A formatação correta do preço



# Testes Funcionais

Download rails40/depot\_e/test/controllers/store\_controller\_test.rb

```
require 'test_helper'
```

```
class StoreControllerTest < ActionController::TestCase
```

```
  test "should get index" do
```

```
    get :index
```

```
    assert_response :success
```

```
➤    assert_select '#columns #side a', minimum: 4
```

```
➤    assert_select '#main .entry', 3
```

```
➤    assert_select 'h3', 'Programming Ruby 1.9'
```

```
➤    assert_select '.price', /\$[, \d]+\.\d\d/
```

```
  end
```

```
end
```

# Testes Funcionais

- As quatro linhas inseridas são responsáveis por investigar o HTML resultante, utilizando a notação de seletor CSS
- Prefixos de seletores
  - # → casa com o id dos atributos (#columns)
  - . → casa com atributos de classe (.price)
  - Nenhum → nome de elementos (h3)
- Para testar – necessários alguns dados

```
# Read about fixtures at  
# http://api.rubyonrails.org/classes/ActiveRecord/Fixtures.html  
one:
```

```
  title: MyString  
  description: MyText  
  image_url: MyString  
  price: 9.99
```

```
two:
```

```
  title: MyString  
  description: MyText  
  image_url: MyString  
  price: 9.99
```

```
ruby:
```

```
  title:      Programming Ruby 1.9  
  description:  
    Ruby is the fastest growing and most exciting dynamic  
    language out there.  If you need to get working programs  
    delivered fast, you should add Ruby to your toolbox.  
  price:      49.50  
  image_url:  ruby.png
```

# Testes Funcionais

- Consulte a documentação para compreender tudo o que um **assert\_select()** pode fazer
- Depois de criados os novos testes, execute-os

```
depot> rake test:controllers
```

– Todos, ou especificamente dos controladores

Iteração C5:

# **ARMAZENANDO RESULTADOS PARCIAIS**

# Feedback do Cliente

- O catálogo será a parte do sistema que será mais utilizadas pelos consumidores
  - Opss! E quando tivermos um número considerável de produtos?
  - Temos como garantir a máxima performance?
  - Será que é necessário recuperar do banco todos os produtos a cada requisição?

# Habilitando a *Cache*

- 1º passo para resolver o problema: habilitar o armazenamento temporário do ambiente

```
Download rails40/depot_e/config/environments/development.rb  
config.action_controller.perform_caching = true
```

- Analisando o próximo passo
  - Só é necessário gerar novamente a página (*render*) se um produto sofrer modificação
  - E gerar novamente apenas o que mudou

# Selecionando o que Recuperar

- Recuperando apenas os produtos recentemente alterados

```
Download rails40/depot_e/app/models/product.rb  
def self.latest  
  Product.order(:updated_at).last  
end
```

- É necessário marcar no *template* os pontos a serem alteradas se um produto for alterado



Download rails40/depot\_e/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>
       </div>
     </div>
  <% end %>
<% end %>
> <% end %>
```

# Trabalhando com *Cache*

- Observações:
  - O gerenciamento da cache fica com o Rails
    - Armazenar os itens e quando invalidar entradas
  - As seções que usam a cache foram demarcadas
    - Encontram-se aninhadas (*matrioska chaching*)
  - Foram identificadas a *cache* geral e cada entrada
    - Foi definida a forma de alimentação da *cache*
  - Desligue a cache para ver as mudanças no *template*
- Veja mais sobre cache nos guias sobre Rails