Programación con Ruby on Rails

Tienda (VI)

Delio Tolivia

Gijón crece 2016 / 2019
AYTO. DE GIJÓN / CCOO / FADE / UGT
Ayuntamiento de Gijón
Gijón impulsa Empleo
dicampus TALENTO FORMACIÓN

Indice

Moviendo el carrito
Añadiendo Ajax
Ocultando el carrito
Haciendo imágenes clickables

Moviendo el carrito (I)

- Vamos a colocar nuestro carrito en la barra lateral. Para ello utilizaremos lo que en rails se denomina “partial templates”. Son trozos de una vista que podemos invocar desde otra plantilla, y podemos pasarle parámetros para que muestre resultados distintos. Empezamos modificando app/views/carts/show.html.erb

```
<% if notice %>
<p id="notice"><%= notice %></p>
<% end %>
<h2>Your Cart</h2>
<table>
<%= render(@cart.line_items) %> #Modificamos la parte de lineas por esto
<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,
   data: {confirm: 'Are you sure'} %>
```

Moviendo el carrito (II)

- El método ***render()*** iterara sobre la colección y utilizará el “partial template” que le llame como la tabla con un “_” delante app/views/line_items/_line_item.html.erb

```
<tr>s
  <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>
```

Moviendo el carrito (III)

- Vamos a crear otra parcial para el carrito app/views/cart/_cart.html.erb

```
<h2>Your Cart</h2>
<table>
<%= render(cart.line_items) %> #Podemos llamar otras parciales desde ella
<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,
   data: {confirm: 'Are you sure'} %>
```

Moviendo el carrito (IV)

- Modificamos nuestra vista del carrito app/views/carts/show.html.erb

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

Moviendo el carrito (V)

- Y modificamos la vista de la aplicación (app/views/layouts/application.html.erb) para incluir el carrito.

```
.........
div id="columns">
      <div id="side">
        <div id="cart">
          <%= render @cart %>
        </div>
        <ul>
          <li><a href="">Home</a></li>
.............
```

Moviendo el carrito (VI)

- Estamos viendo el carrito desde el index donde no se inicializa, por lo que tenemos que modificar el controlador app/controllers/store_controller.erb usando el concern

```
class StoreController < ApplicationController
  include CurrentCart
  before_action :set_cart
  def index
    @products = Product.order(:title)
  end
end
```

Moviendo el carrito (VII)

- Nos queda modificar los css. En app/assets/stylesheets/carts.css.scss

```
.carts, #side #cart{
  .item_price, .total_line{
    text-align: right;
  }
  .total_line, .total_cell{
    font-weight: bold;
    border-top: 1px solid #595;
  }
}
```

Moviendo el carrito (VIII)

- Nos quedaría modificar el comportamiento del botón de añadir al carrito pues ahora nos lleva a una nueva pagina y debería refrescar el index. Para ello modificamos app/controllers/line_items_controller.erb

```
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 } #redirijimos al index
      format.json { render :show, status: :created, location: @line_item }
    else
      format.html { render :new }
      format.json { render json: @line_item.errors, status: :unprocessable_entity }
    end
  end
end
```

Moviendo el carrito (IX)

- Lo mismo ocurre con el botón de vaciar al carrito pues ahora nos lleva a una nueva pagina y debería refrescar el index. Para ello modificamos app/controllers/ carts_controller.erb

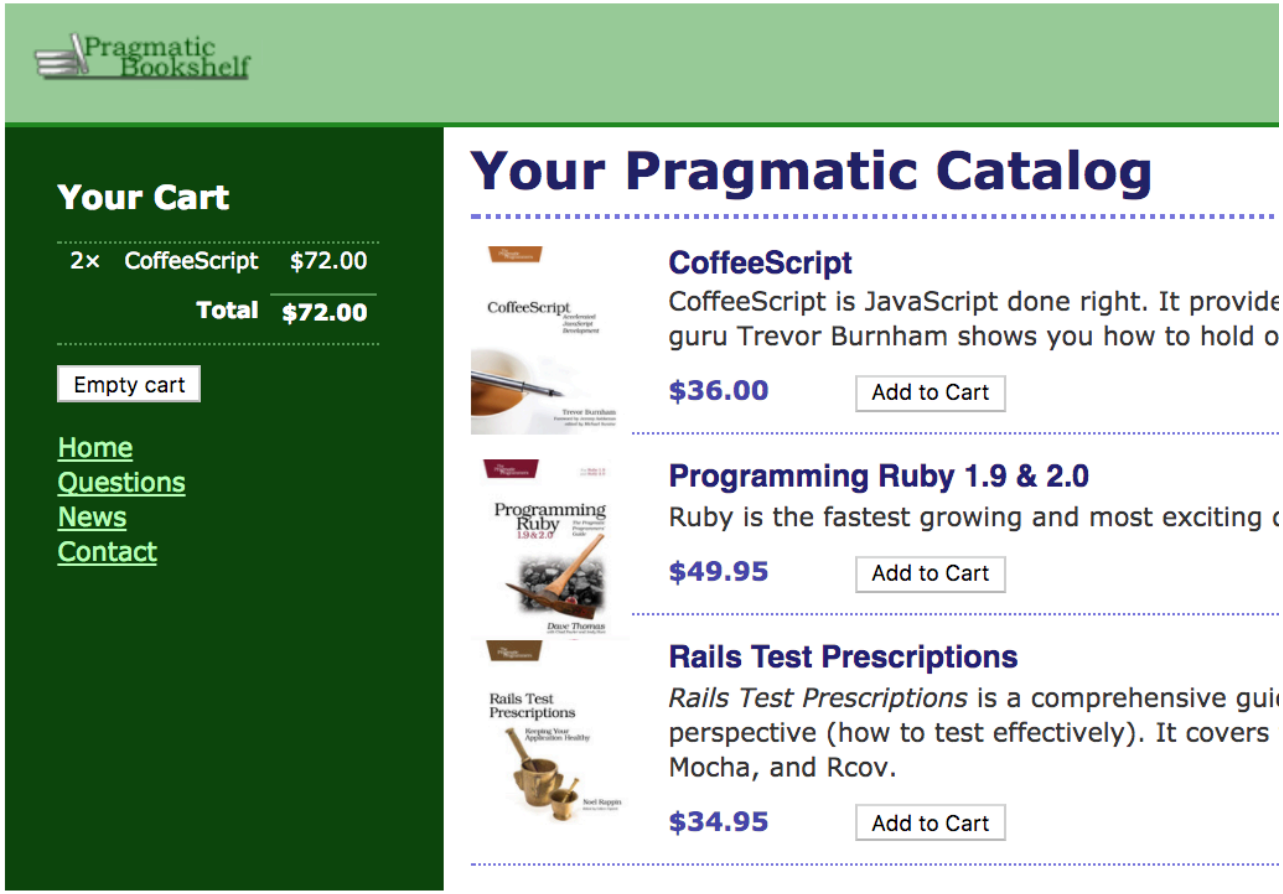
```
def destroy
  @cart.destroy if @cart.id == session[:cart_id]
  session[:cart_id] = nil
  respond_to do |format|
    format.html { redirect_to store_url, notice: 'Your cart is currently empty.' }
    format.json { head :no_content }
  end
end
```

Moviendo el carrito (X)

- Estos cambios producirían un error si intentamos ver los productos y aún no hay un carrito (estaría a nil). Por lo tanto tenemos que corregir nuestra plantilla para que compruebe eso. Modificamos app/views/layouts/application.html.erb

```
<% if @cart %>
  <div id="cart">
    <%=render @cart %>
  </div>
<% end %>
```

Moviendo el carrito (XI)



Añadiendo Ajax (I)

- Ajax, acrónimo de Asynchronous Javascript And XML, es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.

Añadiendo Ajax (II)

- La idea es usar Ajax cuando hacemos click en el botón de añadir al carrito. Para ello modificamos el botón en app/views/store/index.html.erb

```
<%= button_to 'Add to Cart', line_items_path(product_id: product), remote: true %>
```

- Con ese simple cambio del parametro remote: true hacemos que rails ese botón lo asocie a una llamada Ajax

Añadiendo Ajax (III)

- Lo primero es modificar la respuesta de la acción create (app/controllers/line_items_controller.rb) indicando que puede contestar en formato .js

```
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 #Añadimos esto
      format.json { render :show, status: :created, location: @line_item }
    else
      format.html { render :new }
      format.json { render json: @line_item.errors, status: :unprocessable_entity }
    end
  end
end
```

Añadiendo Ajax (IV)

- Con el cambio anterior Rails buscará una plantilla para la acción ***create***. Rails soporta plantillas que generan javascript (.js.erb o .js.coffee). Crearemos la plantilla en app/views/line_items/create.js.coffee (formato coffescript)

cart = document.getElementById("cart")
cart.innerHTML = "<%= j render(@cart) %>"

- Ese script busca el elemento que tiene un ***id="cart"*** sobre el que cambia su contenido html por lo que produce ***render()*** sobre ***@cart***. El método helper ***j()*** convierte la cadena de Ruby en un formato aceptable para el Javascript.

Ocultando carrito vacío

- Vamos a ocultar el carrito cuando está vacío modificando el template parcial del carrito app/views/carts/_cart.html.erb

```
<% unless cart.line_items.empty? %>
<h2>Your Cart</h2>
<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,
   data: {confirm: 'Are you sure'} %>
<% end %>
```

Haciendo imágenes clickables (I)

- Vamos a hacer que se pueda añadir los productos haciendo click en las imágenes. Esto tenemos que hacerlo con javascript mediante el evento onClick de la imagen. Utilizaremos CoffeeScript que simplifica la escritura de código javascript. Y sintaxis jQuery ($). Modificamos el fichero app/assets/javascripts/store.js.coffee

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

Haciendo imágenes clickables (II)

- A partir de Rails 5 no incluye por defecto jquery por lo que tenemos que importarlo en nuestro proyecto. Para ello debemos editar el fichero Gemfile de nuestro proyecto y añadir como primera gema:

gem 'jquery-rails'

- A continuación debemos indicar que queremos utilizarlo en todas nuestras páginas. Para ello editamos el fichero app/assets/javascripts/application.js y añadimos el requerimiento de jquery como primer elemento:

//= require jquery

Haciendo imágenes clickables (III)

- Finalmente tenemos que descargarnos la gema de jquery e instalarla. Para ello desde la carpeta de nuestro proyecto y en la consola ejecutamos:

bundle install

- Después de un rato podemos ver todas las gemas que hay instaladas en nuestro proyecto y comprobar como se descargo e instaló correctamente jquery. A partir de ese momento nuestras imágenes deberán funcionar para añadir productos al carrito.