

Programación con Ruby on Rails

Tienda (IV)

Delio Tolivia





Indice

Creando un carrito

Guardando el id en sesión

Conectando los productos y el carrito

Colocando el botón de añadir al carrito

Resultado

Creando un carrito

- Todas las tiendas online disponen de un carrito donde el usuario va añadiendo sus productos. Ese carrito será almacenado en la base de datos para poder mostrárselo al usuario cuando vuelva si no termino su compra. Para llevar el carrito por las distintas páginas guardaremos en sesión el identificador del carrito ***cart.id***
- Para generar la estructura del carrito ejecutaremos:

rails generate scaffold Cart

rake db:migrate

Guardando el id en sesión

- Para guardar el identificador del carrito en sesión vamos a utilizar los **concerns**. Son una herramienta para poder crear módulos que luego incluiremos en nuestras clases haciendo mixins. En este caso utilizaremos **app/controllers/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
```

Conectando los productos y el carrito (I)

- Nuestro carrito tendrá diferentes líneas de productos por lo que podemos generar el modelo para ellas

```
rails generate scaffold LineItem product:references cart:belongs_to  
rake db:migrate
```

- Si abrimos nuestro nuevo modelo app/models/line_item.rb

```
class LineItem < ApplicationRecord  
  belongs_to :product  
  belongs_to :cart  
end
```

- Nos aparece la relación con producto y carrito. Esto nos permitirá hacer

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

Conectando los productos y el carrito (II)

- Ahora tenemos que implementar la relación inversa. Nuestro carrito tendrá muchas líneas por lo tanto editamos nuestro modelo `app/models/cart.rb`

```
class Cart < ApplicationRecord  
  has_many :line_items, dependent: :destroy  
end
```

- Así ya esta implementada la relación e indicamos que si borramos el carrito también se eliminarán las líneas. Así ya podemos hacer

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

Conectando los productos y el carrito (III)

- Finalmente con el producto `app/models/product.rb`

```
class Product < ApplicationRecord  
  has_many :line_items  
  before_destroy :ensure_not_referenced_by_any_line_item  
  
  #.....  
  
  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
```

- Indicamos la relación y utilizamos un método hook (son métodos que se lanzan automáticamente en algún momento de la vida de un objeto) para asegurarnos que no hay ninguna línea referenciando un producto cuando queremos borrarlo

Colocando el botón de añadir al carro (I)

- Si miramos los métodos que genera nuestro scaffold tenemos ***index()***, ***show()***, ***new()***, ***edit()***, ***create()***, ***update()*** y ***destroy()***. Para nuestro botón nos servirá el método ***create()*** (el método ***new()*** es para mostrar un formulario y rellenar los datos para luego llamar a ***create()***).
- Cuando demos el botón crearemos una ***LineItem***. Si miramos nuestro controlador `app/controllers/line_items_controller.rb`

```
# POST /line_items  
# POST /line_items.json  
def create  
.....
```

- Vemos que nos indica que la url será `/line_items` y el método HTTP será POST

Colocando el botón de añadir al carro (II)

- Para colocar el botón en la vista editaremos app/views/store/index.html.erb


.....

```
<span class="price"><%= number_to_currency(product.price) %></span>  
  <%= button_to 'Add to Cart', line_items_path(product_id: product) %> #esta linea  
</div>
```

.....

- El método button_to añade el form con el botón.

Your Pragmatic Catalog



CoffeeScript
Accelerated
JavaScript
Development

Trevor Burnham
Foreword by Martin Fowler
Edited by Michael Stutz

CoffeeScript

CoffeeScript is JavaScript done right. It's an exciting new language, CoffeeScript gives you cleaner, and safer code.

\$36.00

Add to Cart

Colocando el botón de añadir al carro (III)

- Nos queda modificar el método **create()** de **LineItem** para que reciba un identificador del producto.
- Primero vamos a modificar el controlador (app/controllers/line_items_controller.rb) para que encuentre el carrito de la sesión. Para ellos vamos a hacer uso del concern que realizamos anteriormente

```
class LineItemsController < ApplicationController  
  include CurrentCart  
  before_action :set_cart, only: [:create]  
  .....  
end
```

- Incluimos nuestro modulo del concern e indicamos que el método **:set_cart()** se lance antes del método **create()**

Colocando el botón de añadir al carro (IV)

- Ahora modificamos nuestro create() en el mismo controlador

def create

product = Product.find(params[:product_id]) #Recogemos el producto

@line_item = @cart.line_items.build(product: product) #Creamos la linea en el carrito

respond_to do |format|

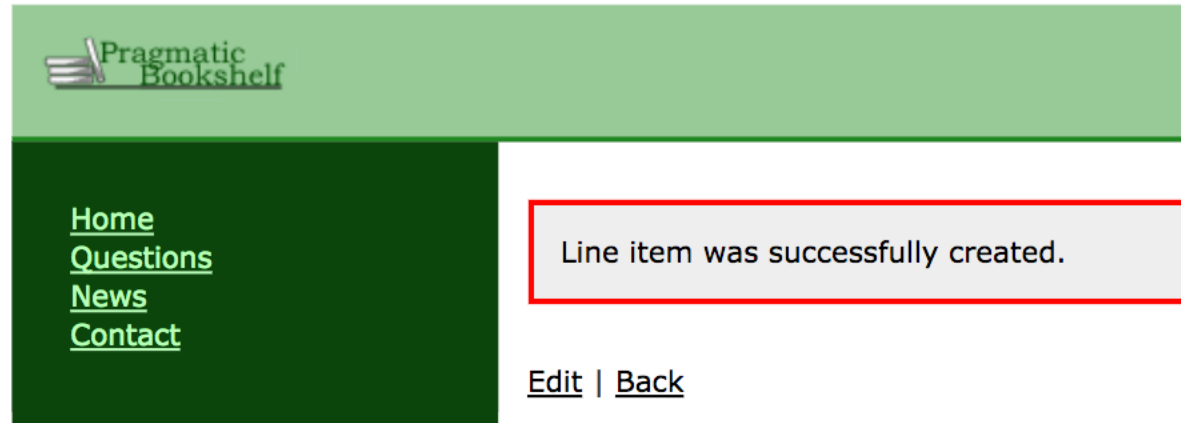
if @line_item.save

format.html { redirect_to @line_item.cart, notice: 'Line item was successfully created.' } #redirigimos al carrito

.....

Colocando el botón de añadir al carro (V)

- Ya podemos utilizar nuestro botón de añadir al carrito




- Como el carrito no tiene ningún atributo la vista no muestra nada. Tenemos que modificarla

Colocando el botón de añadir al carro (VI)

- Modificamos la vista 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>
```

Resultado



PRAGMATIC BOOKSHELF

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

Line item was successfully created.

Your Pragmatic Cart

- CoffeeScript
- Programming Ruby 1.9 & 2.0
- CoffeeScript