

Programación con Ruby on Rails

Tienda (V)

Delio Tolivia





Indice

Mejorando el carrito

Manejando errores

Vaciando el carrito

Terminando el carrito

Resumen

Mejorando el carrito (I)

- Nuestro carrito tiene un problema. Si añadimos el mismo producto más de una vez nos repite las líneas por lo que tenemos que modificar nuestro **LineItem** para añadirle un atributo de cantidad del producto.
- Para modificarla ejecutaremos

rails generate migration add_quantity_to_line_items quantity:integer

- Rails busca **add_XXX_to_TABLE** o **remove_XXX_from_TABLE** para añadir columnas o eliminarlas.
- Esto nos crea el fichero db/migrate/XXXXXXXXXXXX_add_quantity_to_line_items.rb

Mejorando el carrito (II)

- Editamos el fichero de migración

```
class AddQuantityToLineItems < ActiveRecord::Migration[5.1]  
def change  
  add_column :line_items, :quantity, :integer, default:1  
end  
End
```

- Añadimos para que por defecto le de el valor 1 al contador
- Para ejecutar la migración ejecutamos ***rake db:migrate***

Mejorando el carrito (III)

- Ahora vamos a modificar el modelo del carrito (app/models/cart.rb) para que tenga un método ***add_product()*** que compruebe si ya existe el producto en una línea y aumente su cantidad

```
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
```

Mejorando el carrito (IV)

- Tenemos que modificar el controlador de LineItems (app/controllers/line_items_controller.rb) para que lo utilice

def create

product = Product.find(params[:product_id])

@line_item = @cart.add_product(product.id)

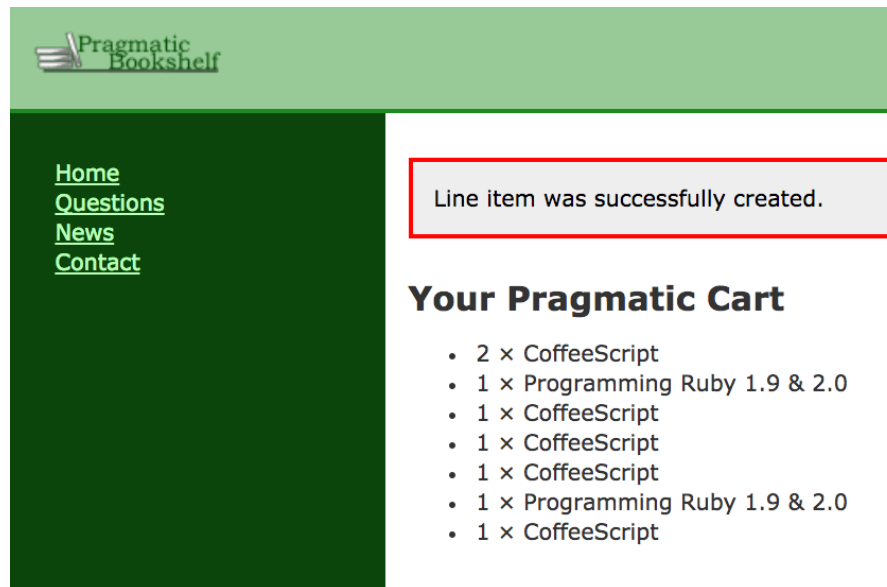
respond_to do |format|

.....

Mejorando el carrito (V)

- Finalmente cambiamos la vista para mostrarlo (app/views/carts/show.html.erb)

`<%= item.quantity%> × <%= item.product.title %> `



Mejorando el carrito (VI)

- Vemos que ya funciona pero los elementos que había ya en el carrito nos aparecen con un solo elemento. Esto podemos solucionarlos con una migración para que los acumule.

rails generate migration combine_items_in_cart

- Ahora tenemos que escribir el método que hace falta para que los acumule pues rails no sabe como hacerlo. Ese método se llama **up()**, el contrario que permite hacer un rollback se llama **down()**. Editamos el fichero db/migrate/XXXXXXXX_combine_items_in_cart

Mejorando el carrito (VI)

def up

#replace multiple items for a single product in a cart with a single item

Cart.all.each do |cart|

#count the number of each product in the cart

sums = cart.line_items.group(:product_id).sum(:quantity)

sums.each do |product_id, quantity|

if quantity>1

#remove individual items

cart.line_items.where(product_id: product_id).delete_all

#replace with a single line

item = cart.line_items.build(product_id: product_id)

item.quantity = quantity

item.save!

end

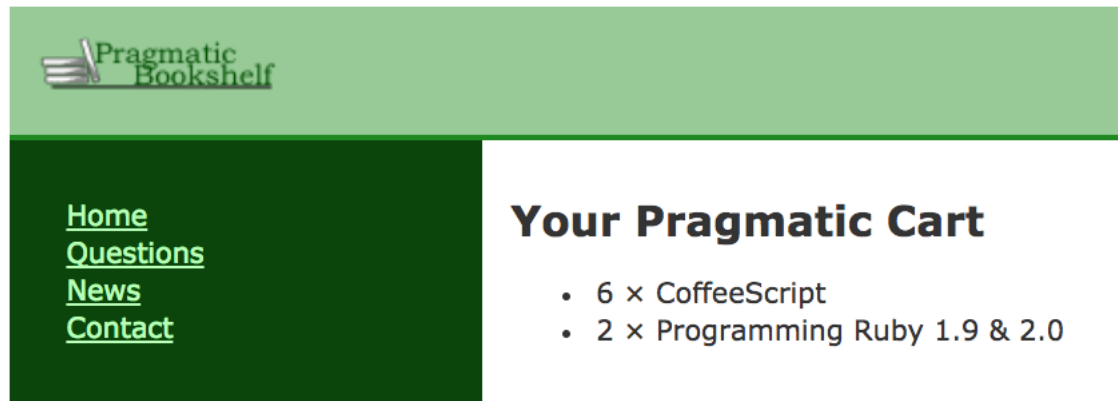
end

end

end

Mejorando el carrito (VII)

- Una vez terminado podemos hacer `rake db:migrate` y ya actualizará nuestra base de datos



- Con ***rake db:status*** podemos ver el estado de nuestras migraciones y ***rake db:rollback*** cuando queremos eliminar los cambios hechos por una (tendríamos que tener el método ***down()***)

Manejando errores (I)

- Si visitamos la página <http://localhost:3000/carts/wibble>

ActiveRecord::RecordNotFound in CartsController#show

Couldn't find Cart with 'id'=wibble

Extracted source (around line #67):

```
65     # Use callbacks to share common setup or constraints between actions.
66     def set_cart
67       @cart = Cart.find(params[:id])
68     end
69
70     # Never trust parameters from the scary internet, only allow the white list through.
```

- Hay un error cuando nos introducen en la url una id que no existe y además con ello se puede ver parte de nuestra implementación

Manejando errores (II)

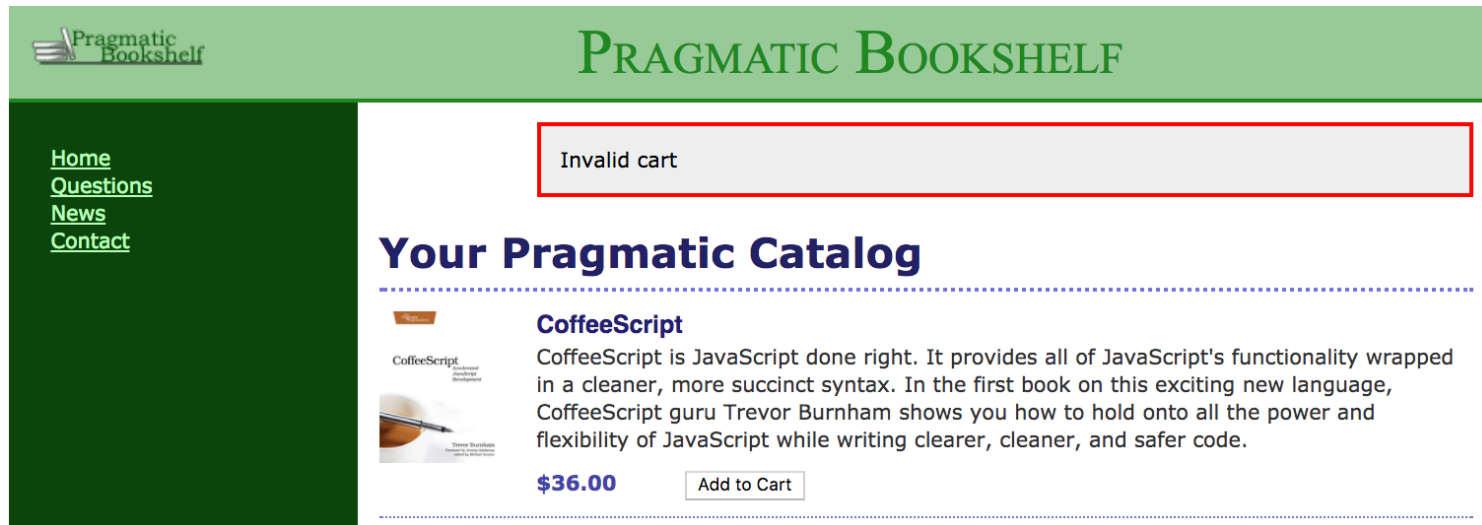
- Rails nos proporciona una herramienta llamada flash que es similar a un hash donde podemos almacenar valores que estarán disponibles en la siguiente request. Normalmente se utiliza para almacenar errores. En nuestro caso almacenaremos el error cuando el método show() detecta un error y redirigimos al index que extraerá el error.
- Definiremos un método invalid_cart() que implementaremos en nuestro controlador app/controllers/carts_controller.rb

Manejando errores (III)

```
class CartsController < ApplicationController  
  before_action :set_cart, only: [:show, :edit, :update, :destroy]  
  rescue_from ActiveRecord::RecordNotFound, with: :invalid_cart  
  # GET /carts  
  # GET /carts.json  
  
  .....  
  def invalid_cart  
    logger.error "Attemp to access invalid cart #{params[:id]}"  
    redirect_to store_url, notice: 'Invalid cart'  
  end
```

Manejando errores (IV)

- Si volvemos a intentar introducir la dirección incorrecta nos redirige al index



- Además podemos mirar en el log (/log/development.log) y veremos en la parte de abajo:

Attemp to access invalid cart wibble

Vaciando el carrito (I)

- Lo primero que haremos será modificar la vista (app/views/carts/show.html.erb) para que nos aparezca un botón para vaciarlo.

```
<% if notice %>
<p id="notice"><%= notice %></p>
<% end %>
<h2>Your Pragmatic Cart</h2>
<ul>
  <% @cart.line_items.each do |item| %>
    <li><%= item.quantity%> &times; <%= item.product.title %> </li>
  <% end %>
</ul>
<%= button_to 'Empty cart', @cart, method: :delete,
  data: {confirm: 'Are you sure'} %>
```

Vaciando el carrito (II)

- Por lo tanto tenemos que actualizar nuestro método ***destroy()*** del controlador (app/controllers/carts_controller.erb) para asegurarnos que elimina el carrito asociado a la sesión del usuario

def destroy

@cart.destroy if @cart.id == session[:cart_id]

session[:cart_id] = nil

respond_to do |format|

format.html { redirect_to carts_url, notice: 'Your cart is currently empty.' }

format.json { head :no_content }

end

end

- Con eso ya aparece el botón y podemos vaciar el carrito.

Terminando el carrito (I)

- Ya podemos eliminar el mensaje que nos generaba LineItem cada vez que se generaba una nueva linea en app/controllers/line_items_controller.rb

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 @line_item.cart } #Quitamos el mensaje de aqui

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

Terminando el carrito (II)

- Vamos a mejorar la vista de nuestro carrito (app/views/carts/show.html.erb)

```
<% if notice %>
<p id="notice"><%= notice %></p>
<% end %>
<h2>Your Cart</h2>
<table>
  <% @cart.line_items.each do |item| %>
    <tr>
      <td><%= item.quantity%>&times;</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,
  data: {confirm: 'Are you sure'} %>
```

Terminando el carrito (III)

- Hemos utilizado un método ***total_price()*** que no tenemos definido ni en `LineItem`(app/models/line_item.rb) ni en `Cart` (app/models/cart.rb).

LineItem

```
def total_price  
  product.price * quantity  
end
```

Cart


```
def total_price  
  line_items.to_a.sum {|item| item.total_price}  
end
```

Terminando el carrito (IV)

- Por ultimo añadimos unas reglas de css en app/assets/stylesheets/cart.css.scss

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

Resultado



PRAGMATIC BOOKSHELF

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

Your Cart

2× CoffeeScript `number_to_currency(item.total_price) %>`

Total	\$72.00
--------------	----------------

Empty cart