Web Engineering:
Task: A smarter cart

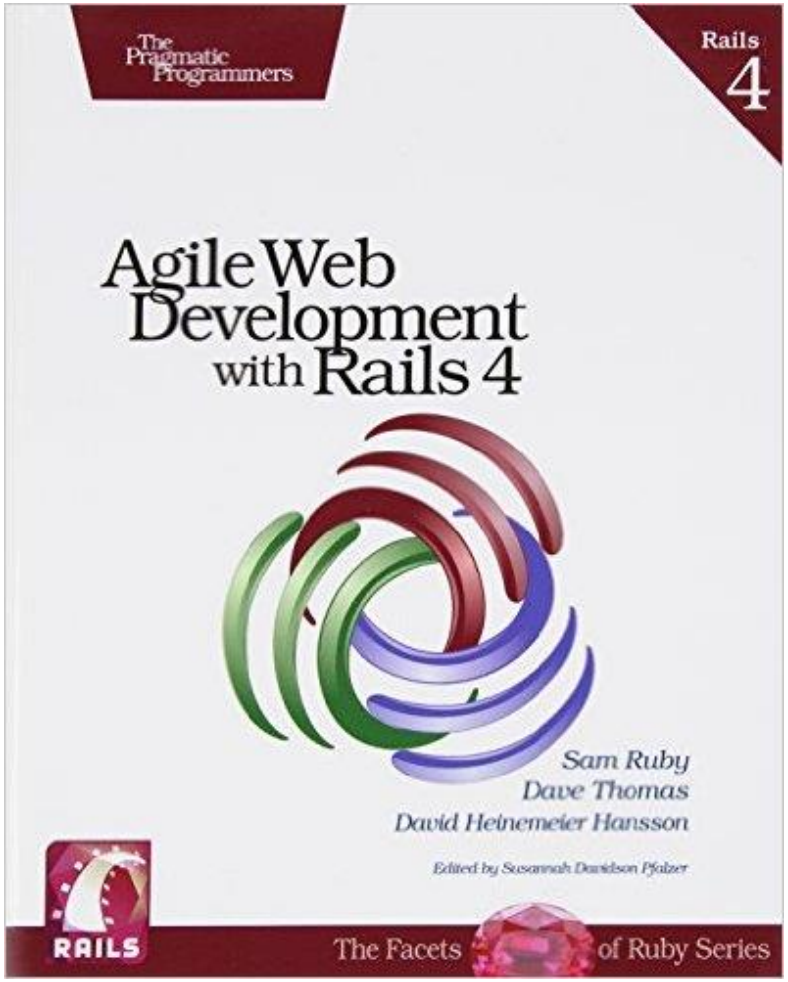
The University of Aizu
Quarter 2, AY 2018

Outline

- Creating a smarter cart
- Handling errors
- Finishing the cart

Literature

- Agile Web Development with Rails 4 (1st edition) by Sam Ruby, Dave Thomas and Devid Hansson, The Pragmatic Bookshelf, 2013.
  - Chapter 10.



Creating a smarter cart

- We have rudimentary cart functionality implemented.
- Points to improve:
  - We need to recognize when customers add multiples of the same item to the cart.
  - The cart itself can handle error cases.

Creating a smarter cart

- Associating a count with each product in our cart is going to require us to modify the *line_items* table.

```
depot> rails generate migration add_quantity_to_line_items quantity:interger
```

- Rails can tell from the name of migration that you are adding one or more columns to the *line_items* table and can pick up the names and data types for each column from the last argument.
  - The patterns that Rails matches on are *add_XXX_to_TABLE* and *remove_XXX_from_TABLE*
- Rails cannot tell the default value. The developer need to do this as follows (next slide).

Creating a smarter cart

Download rails40/depot_g/db/migrate/20121130000004_add_quantity_to_line_items.rb

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

- After that, we need to run:

depot> rake db:migrate

Creating a smarter cart

- Now we need to add a smart *add_product* method to our *Cart*:
  - If the list of items is already includes the product, it increase the quantity by 1;
  - If the list of items does not include the product, it creates a new *LineItem*.

Download rails40/depot_g/app/models/cart.rb

```
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
```

Comments on the previous slide

- The *find_by* method returns either an existing *LineItem* or *nil*.
- We need to modify line item controller to make use of this method (see the next slide).

Creating a smarter cart

Download rails40/depot_g/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,
        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
```

Creating a smarter cart

- The view should be adjusted to use this information:

Download rails40/depot_g/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.quantity %> &times; <%= item.product.title %></li>
  <% end %>
</ul>
```

- Now, we need a migration:

depot> rails generate migration combine_items_in_cart

Creating a smarter cart

- Rails cannot infer that we are trying to do, so we to introduce separate *up* and *down* methods.

Download rails40/depot_g/db/migrate/20121130000005_combine_items_in_cart.rb

```
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 item
        item = cart.line_items.build(product_id: product_id)
        item.quantity = quantity
        item.save!
      end
    end
  end
end
```

Comments on the previous slide

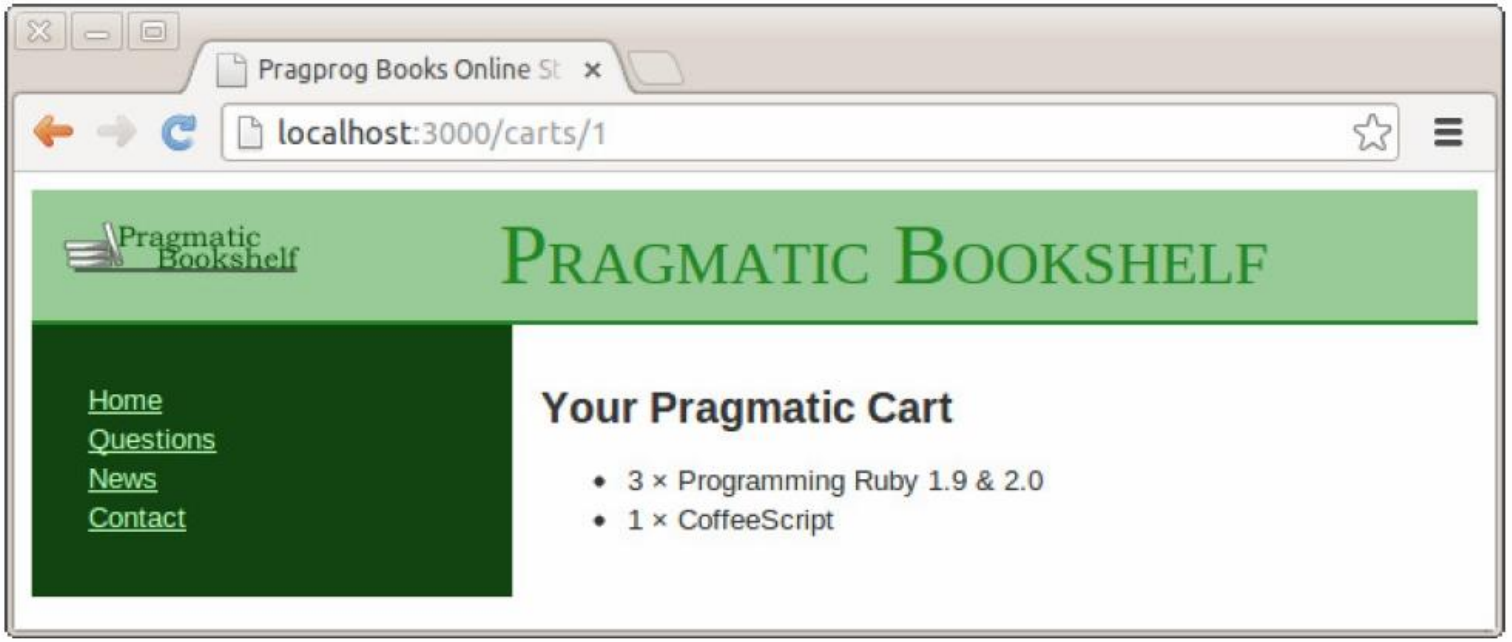
- We start by iterating over each cart
- For each cart, we get a sum of the quantity fields for each of the line items associated with this cart, groped by *product_id*. The resulting sums will be a list of ordered pairs of *product_ids* and quantity.
- We iterate over these sums, extracting the *product_id* and *quantity* from each.
- In cases where the quantity is greater than 1, we will delete all of the individual line items associated with this cart and this product and replace them with a single line item with correct quantity.

Creating a smarter cart

- Now, we should apply this migration:

  depot> rake db:migrate

- We can immediately see the result:



Creating a smarter cart

- An important principle of migrations is that each step needs to be reversible.
- We implement a *down* too (see the next slide):
  - This method finds line items with the quantity of greater than 1.
  - It adds new line items for this cart and product, each with a quantity of 1.
  - It finally deletes the line item.

down method

Download rails40/depot_g/db/migrate/20121130000005_combine_items_in_cart.rb

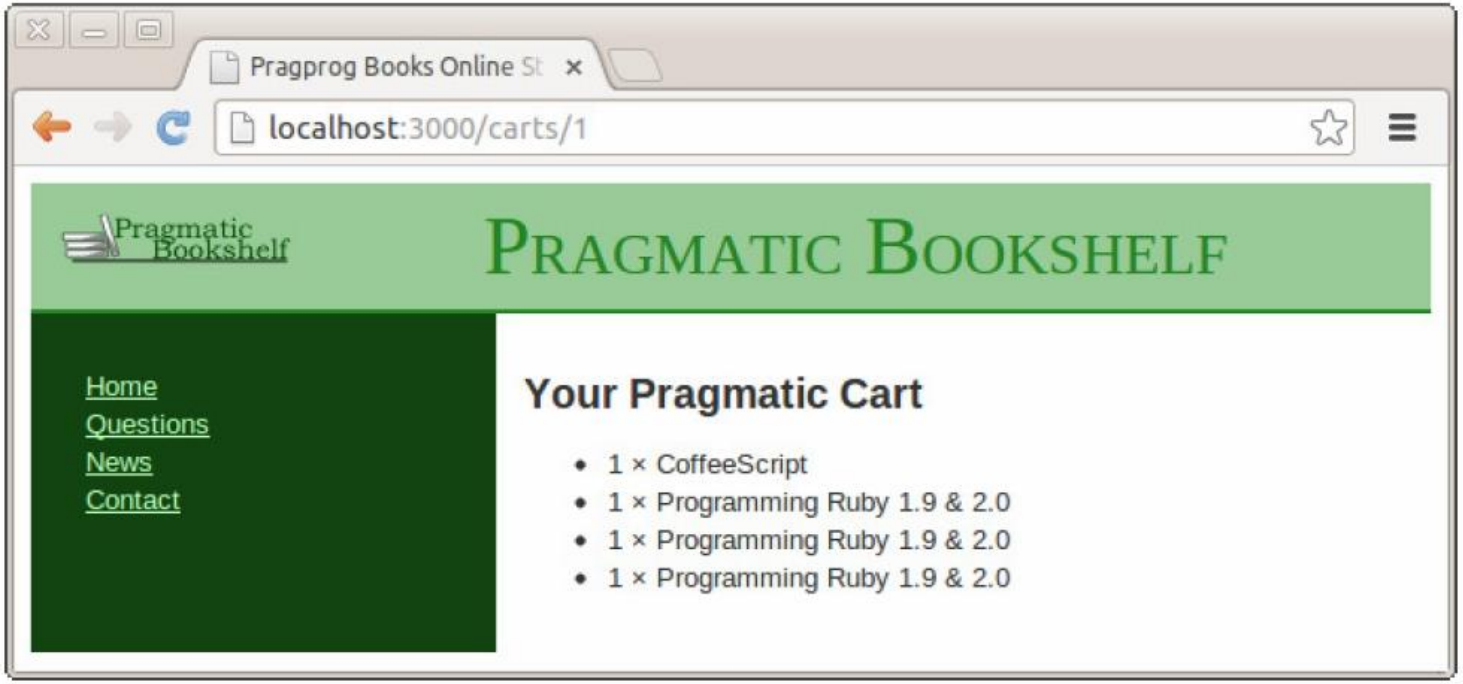
```
def down
  # split items with quantity>1 into multiple items
  LineItem.where("quantity>1").each do |line_item|
    # add individual items
    line_item.quantity.times do
      LineItem.create cart_id: line_item.cart_id,
        product_id: line_item.product_id, quantity: 1
    end

    # remove original item
    line_item.destroy
  end
end
```

Creating a smarter cart

- At this poin, we can easily roll back our migration with a single command:

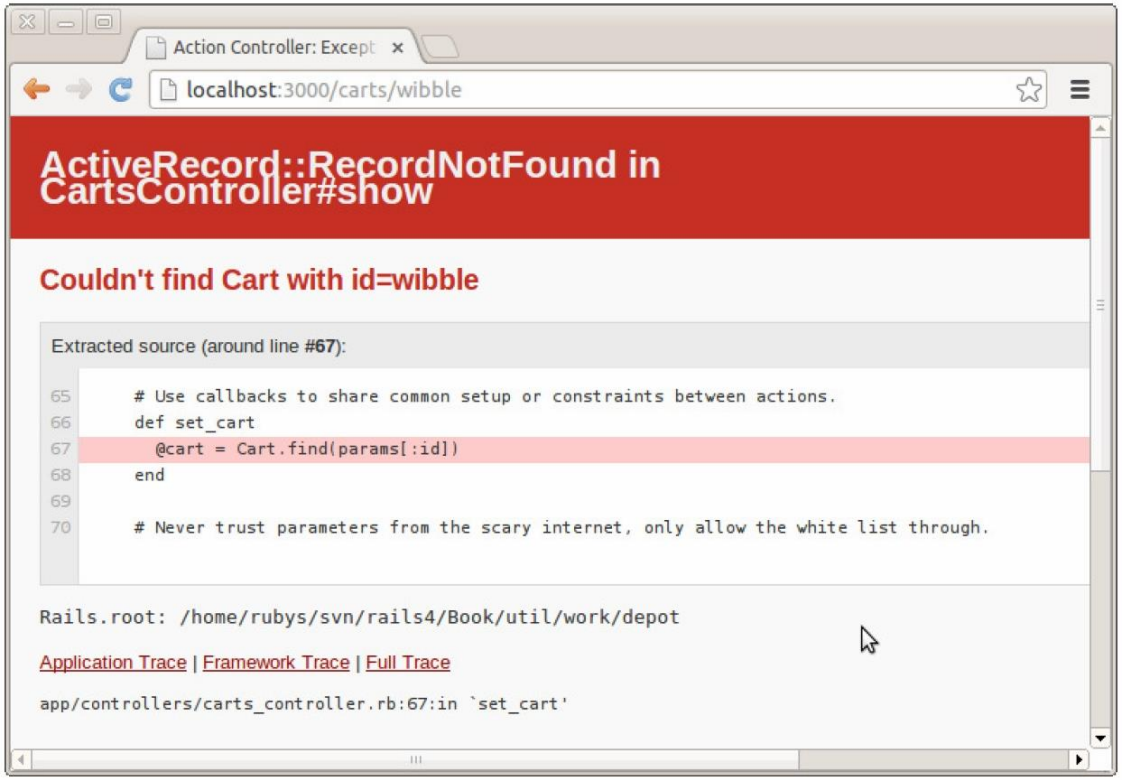
depot> rake db:rollback

- We can immediately inspect the result by looking at the cart:



Creating a smarter cart

- If you will type in the browser something like this: http://localhost:3000/carts/wibble
- The application response is in the figure below:



- Our next goal is to make application more resilient.

Handling errors

- Our application raises an exception at line 67 of the cart controller (see the previous slide):

```
@cart = Cart.find(params[:id])
```

- This situation is bad from the security point of view. To prevent any attacks, we will do:
  - We will log the error fact to an internal log file using Rails logger facilities.
  - We will redisplay the catalog page with a short message to the user (Invalid cart), so they can continue to use our site.
- Rails has a convenient way of dealing with errors. It defines a *flash* structure:
  - This is a bucket there you can store stuff as you process a request.

Handling errors

- Now, we can create an *invalid_cart* method to report on the problem:

Download rails40/depot_h/app/controllers/carts_controller.rb

```
class CartsController < ApplicationController
  before_action :set_cart, only: [:show, :edit, :update, :destroy]
  rescue_from ActiveRecord::RecordNotFound, with: :invalid_cart
  # GET /carts
  # ...
  private
  # ...

    def invalid_cart
      logger.error "Attempt to access invalid cart #{params[:id]}"
      redirect_to store_url, notice: 'Invalid cart'
    end
end
```

Comments on the previous slide

- The *rescue_from* clause intercepts the exception raised by *Cart.find* . In the handler, we do the following:
  - Use the Rails logger to record the error. Every controller has a logger attribute. Here, we use it to record a message at the error logging level.
  - Redirect to the catalog display using the *redirect_to* method. The *:notice* parameter specifies a message to be stored in the flash as a notice.

Handling errors

- With this code in place, the application will response to our wrong URL: http://localhost:3000/carts/wibble

Handling errors

- If you will look at the end of the log file (*development.log* in the *log* directory), you will find our message:

```
Started GET "/carts/wibble" for 127.0.0.1 at 2013-01-29 09:37:39 -0500
Processing by CartsController#show as HTML
  Parameters: {"id"=>"wibble"}
  ^[[1m^[[35mCart Load (0.1ms)^[[0m  SELECT "carts".* FROM "carts" WHERE
"carts"."id" = ? LIMIT 1  [["id", "wibble"]]
➤ Attempt to access invalid cart wibble
Redirected to http://localhost:3000/
Completed 302 Found in 3ms (ActiveRecord: 0.4ms)
```

Finishing the cart

- To implement the "*empty cart*" function, we have to add a link to the cart and modify the *destroy* method in the carts controller to clean up the session.
- We start with the *button_to* method to put a button on the page:

Download rails40/depot_h/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.quantity %> &times; <%= item.product.title %></li>
  <% end %>
</ul>
<%= button_to 'Empty cart', @cart, method: :delete,
    data: { confirm: 'Are you sure?' } %>
```

Finishing the cart

- In the controller, we will modify the *destroy* method to ensure that the user is deleting their own cart and to remove the cart from the session:

Download rails40/depot_h/app/controllers/carts_controller.rb

```
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
```

Finishing the cart

- We update the corresponding test in:

Download rails40/depot_i/test/controllers/carts_controller_test.rb

```
test "should destroy cart" do
  assert_difference('Cart.count', -1) do
➤   session[:cart_id] = @cart.id
    delete :destroy, id: @cart
  end

➤ assert_redirected_to store_path
end
```

Finishing the cart

- We can also remove the *flash* message that is automatically generated when a line item is added:

Download rails40/depot_i/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 }
      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
```

Finishing the cart

- Now, it is time to improve the cart display. We will use table structure and rely on CSS:

Download rails40/depot_i/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?' } %>
```

Finishing the cart

- We need to add a method to both the *LineItem* and *Cart* models that returns the total price for the individual line item and entire cart, respectively.
- First, the line item, which involves only simple multiplication:

Download rails40/depot_i/app/models/line_item.rb

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

Finishing the cart

- We implement the cart method using Rails' nifty *Array::sum* method to sum the prices of each item in the collection.

Download rails40/depot_i/app/models/cart.rb

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

Finishing the cart

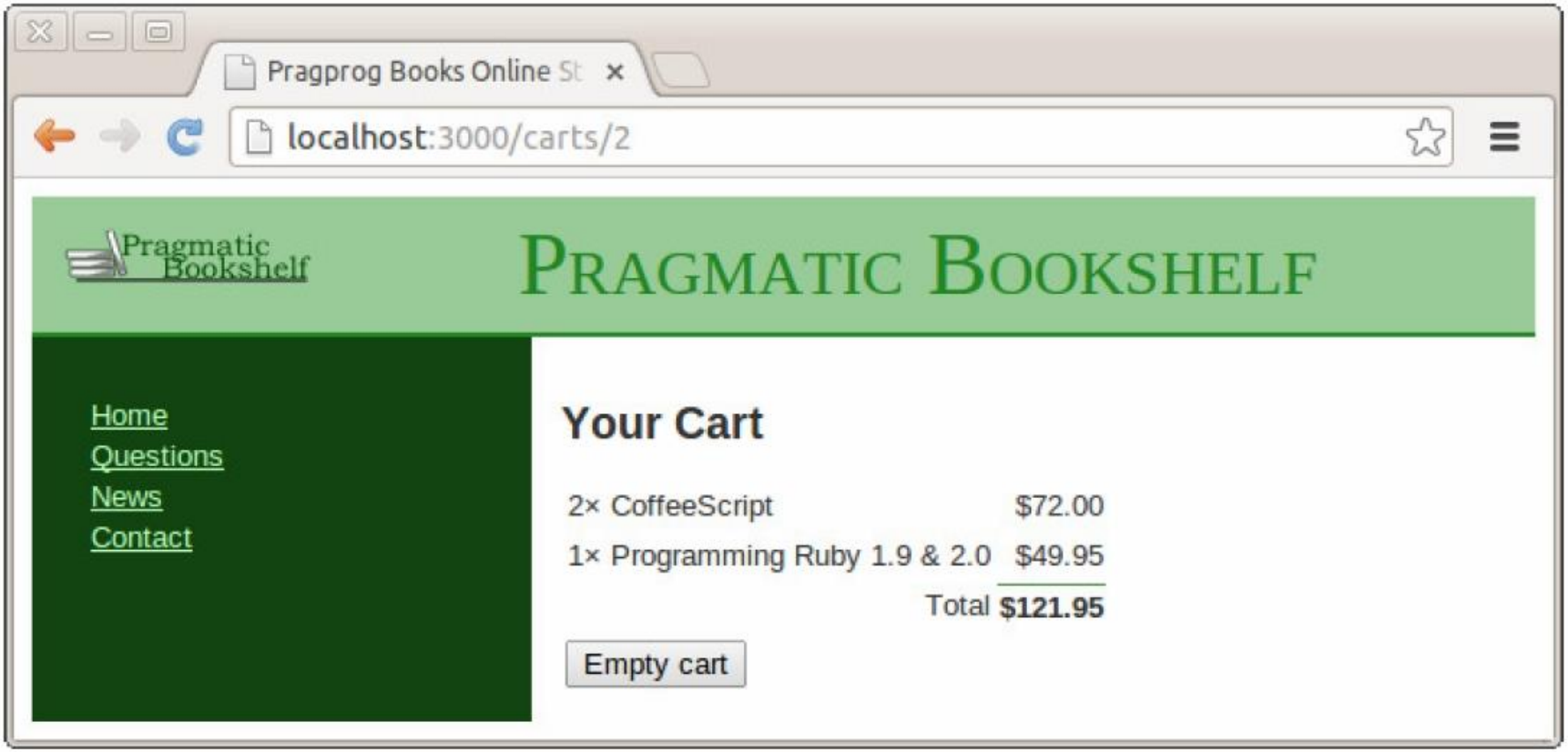
- Finally, we need to add a small piece of code to our carts.css.scss stylesheet:

Download rails40/depot_i/app/assets/stylesheets/carts.css.scss

```
// Place all the styles related to the Carts controller here.
// They will automatically be included in application.css.
// You can use Sass (SCSS) here: http://sass-lang.com/
.carts {
  .item_price, .total_line {
    text-align: right;
  }
  .total_line .total_cell {
    font-weight: bold;
    border-top: 1px solid #595;
  }
}
```

Finishing the cart

- A nice-looking cart is here:



What we just did

- The client likes our shopping cart!
- We covered the following:
  - Adding the column to an existing table, with the default value
  - Migrating existing data into the new table format
  - Producing a flash notice of an error that was detected
  - Using the logger to log events
  - Deleting a record
  - Adjusting the way a table is rendered, using CSS