Extracted from:

Agile Web Development with Rails 5

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Rails 5

Agile Web Development with Rails 5

Sam Ruby
Dave Thomas
David Heinemeier Hansson

Edited by Susannah Davidson Pfalzer

RAILS

Agile Web Development with Rails 5

Sam Ruby
Dave Thomas
David Heinemeier Hansson

The Pragmatic Bookshelf
Raleigh, North Carolina

The team that produced this book includes:

Susannah Davidson Pfalzer (editor)
Potomac Indexing, LLC (index)
Eileen Cohen (copyedit)
Gilson Graphics (layout)
Janet Furlow (producer)

Printed in the United States of America.
ISBN-13: 978-1-68050-171-1
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—September 2016

Iteration D3: Adding a Button

Now that that's done, it's time to add an Add to Cart button for each product.

We don't need to create a new controller or even a new action. Taking a look at the actions provided by the scaffold generator, we find index(), show(), new(), edit(), create(), update(), and destroy(). The one that matches this operation is create(). (new() may sound similar, but its use is to get a form that's used to solicit input for a subsequent create() action.)

Once this decision is made, the rest follows. What are we creating? Certainly not a Cart or even a Product. What we're creating is a LineItem. Looking at the comment associated with the create() method in app/controllers/line_items_controller.rb, you see that this choice also determines the URL to use (/line_items) and the HTTP method (POST).

This choice even suggests the proper UI control to use. When we added links before, we used link_to(), but links default to using HTTP GET. We want to use POST, so we'll add a button this time; this means we'll be using the button_to() method.

We could connect the button to the line item by specifying the URL, but again we can let Rails take care of this for us by simply appending _path to the controller's name. In this case, we'll use line_items_path.

However, there's a problem with this: how will the line_items_path method know *which* product to add to our cart? We'll need to pass it the ID of the product corresponding to the button. That's easy enough—all we need to do is add the :product_id option to the line_items_path() call. We can even pass in the product instance itself—Rails knows to extract the ID from the record in circumstances such as these.

In all, the *one* line that we need to add to our index.html.erb looks like this:

rails50/depot_f/app/views/store/index.html.erb

```
<p id="notice"><%= notice %></p>

<h1>Your Pragmatic Catalog</h1>

<% cache @products do %>
  <% @products.each do |product| %>
    <% cache 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>
          <%= button_to 'Add to Cart', line_items_path(product_id: product) %>
        </div>
      </div>
    <% end %>
  <% end %>
<% end %>
```

We need to deal with one more formatting issue. button_to creates an HTML <form>, and that form contains an HTML <div>. Both of these are normally block elements that appear on the next line. We'd like to place them next to the price, so we need to add a little CSS magic to make them inline:

rails50/depot_f/app/assets/stylesheets/store.scss

```
p, div.price_line {
  margin-left: 100px;
  margin-top: 0.5em;
  margin-bottom: 0.8em;

  form, div {
    display: inline;
  }
}
```

Now our index page looks like the screenshot on page 7. But before we push the button, we need to modify the create() method in the line items controller to expect a product ID as a form parameter. Here's where we start to see how important the id field is in our models. Rails identifies model objects (and the corresponding database rows) by their id fields. If we pass an ID to create(), we're uniquely identifying the product to add.

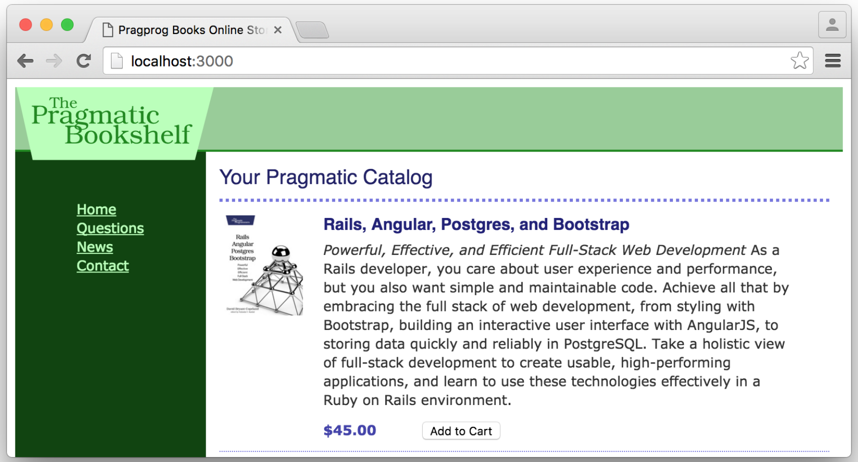
Why the create() method? The default HTTP method for a link is a GET, and for a button is a POST. Rails uses these conventions to determine which method to call. Refer to the comments inside the app/controllers/line_items_controller.rb file to see other conventions. We'll be making extensive use of these conventions inside the Depot application.

Now let's modify the LineItemsController to find the shopping cart for the current session (creating one if one isn't there already), add the selected product to that cart, and display the cart contents.

We use the CurrentCart concern we implemented in Iteration D1 on page ? to find (or create) a cart in the session:

rails50/depot_f/app/controllers/line_items_controller.rb

```
class LineItemsController < ApplicationController
  include CurrentCart
  before_action :set_cart, only: [:create]
  before_action :set_line_item, only: [:show, :edit, :update, :destroy]
```



Your Pragmatic Catalog

Rails, Angular, Postgres, and Bootstrap

Powerful, Effective, and Efficient Full-Stack Web Development As a Rails developer, you care about user experience and performance, but you also want simple and maintainable code. Achieve all that by embracing the full stack of web development, from styling with Bootstrap, building an interactive user interface with AngularJS, to storing data quickly and reliably in PostgreSQL. Take a holistic view of full-stack development to create usable, high-performing applications, and learn to use these technologies effectively in a Ruby on Rails environment.

$45.00

```
  # GET /line_items
  #...
end
```

We include the CurrentCart module and declare that the set_cart() method is to be involved before the create() action. We explore action callbacks in depth in *Callbacks, on page ?*, but for now all you need to know is that Rails provides the ability to wire together methods that are to be called before, after, or even around controller actions.

In fact, as you can see, the generated controller already uses this facility to set the value of the @line_item instance variable before the show(), edit(), update(), or destroy() actions are called.

Now that we know that the value of @cart is set to the value of the current cart, all we need to modify is a few lines of code in the create() method in app/controllers/line_items_controller.rb. to build the line item itself:

rails50/depot_f/app/controllers/line_items_controller.rb

```
  def create
➤   product = Product.find(params[:product_id])
➤   @line_item = @cart.line_items.build(product: product)

    respond_to do |format|
      if @line_item.save
➤       format.html { redirect_to @line_item.cart,
          notice: 'Line item was successfully created.' }
        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
```

We use the params object to get the :product_id parameter from the request. The params object is important inside Rails applications. It holds all of the parameters passed in a browser request. We store the result in a local variable because there's no need to make this available to the view.

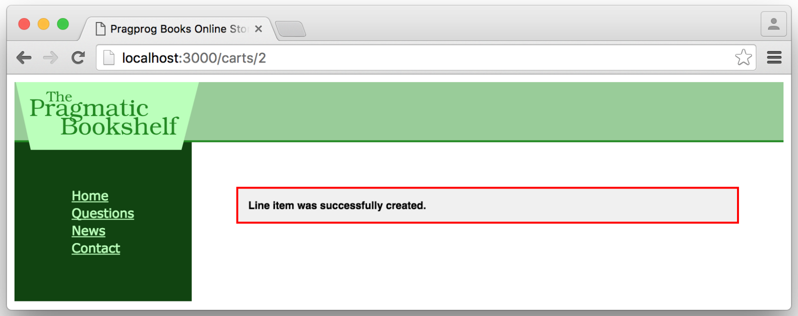
We then pass that product we found into @cart.line_items.build. This causes a new line item relationship to be built between the @cart object and the product. You can build the relationship from either end, and Rails takes care of establishing the connections on both sides.

We save the resulting line item into an instance variable named @line_item.

The remainder of this method takes care of handling errors, which we'll cover in more detail in *Iteration E2: Handling Errors, on page ?*, and handling JSON requests. But for now, we want to modify only one more thing: once the line item is created, we want to redirect users to the cart instead of back to the line item. Since the line item object knows how to find the cart object, all we need to do is add .cart to the method call.

Confident that the code works as intended, we try the Add to Cart buttons in our browser.

And the following screenshot shows what we see.



Line item was successfully created.

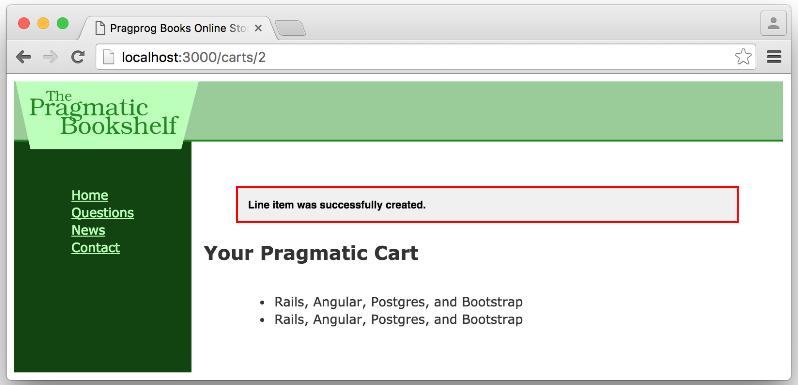
This is a bit underwhelming. We have scaffolding for the cart, but when we created it we didn't provide any attributes, so the view doesn't have anything to show. For now, let's write a trivial template (we'll tart it up in a minute):

rails50/depot_f/app/views/carts/show.html.erb

```
<p id="notice"><%= notice %></p>

<h2>Your Pragmatic Cart</h2>
<ul>
  <% @cart.line_items.each do |item| %>
    <li><%= item.product.title %></li>
  <% end %>
</ul>
```

So, with everything plumbed together, let's go back and click the Add to Cart button again and see our simple view displayed, as in the next screenshot.



Line item was successfully created.

Your Pragmatic Cart

- Rails, Angular, Postgres, and Bootstrap
- Rails, Angular, Postgres, and Bootstrap

Go back to http://localhost:3000/, the main catalog page, and add a different product to the cart. You'll see the original two entries plus our new item in your cart. It looks like we have sessions working.

We changed the function of our controller, so we know that we need to update the corresponding functional test.

For starters, we only need to pass a product ID on the call to post. Next, we have to deal with the fact that we're no longer redirecting to the line items page. We're instead redirecting to the cart, where the cart ID is internal state data residing a cookie. Because this is an integration test, instead of focusing on how the code is implemented, we should focus on what users see after following the redirect: a page with a heading identifying that they're looking at a cart, with a list item corresponding to the product they added.

We do this by updating test/controllers/line_items_controller_test.rb:

rails50/depot_g/test/controllers/line_items_controller_test.rb
```
  test "should create line_item" do
    assert_difference('LineItem.count') do
➤     post line_items_url, params: { product_id: products(:ruby).id }
    end

➤   follow_redirect!
➤
➤   assert_select 'h2', 'Your Pragmatic Cart'
➤   assert_select 'li', 'Programming Ruby 1.9'
  end
```

We now rerun this set of tests:

```
depot> bin/rails test test/controllers/line_items_controller_test.rb
```

It's time to show our customer, so we call her over and proudly display our handsome new cart. Somewhat to our dismay, she makes that *tsk-tsk* sound that customers make just before telling you that you clearly don't get something.

Real shopping carts, she explains, don't show separate lines for two of the same product. Instead, they show the product line once with a quantity of 2. It looks like we're lined up for our next iteration.

What We Just Did

It's been a busy, productive day so far. We added a shopping cart to our store, and along the way we dipped our toes into some neat Rails features:

- We created a Cart object in one request and successfully located the same cart in subsequent requests by using a session object.
- We added a private method and placed it in a concern, making it accessible to all of our controllers.
- We created relationships between carts and line items and relationships between line items and products, and we were able to navigate using these relationships.
- We added a button that causes a product to be posted to a cart, causing a new line item to be created.

Playtime

Here's some stuff to try on your own:

- Add a new variable to the session to record how many times the user has accessed the store controller's index action. Note that the first time this page is accessed, your count won't be in the session. You can test for this with code like this:

  ```
  if session[:counter].nil?
    ...
  ```

  If the session variable isn't there, you need to initialize it. Then you'll be able to increment it.
- Pass this counter to your template, and display it at the top of the catalog page. Hint: the pluralize helper (definition on page ?) might be useful for forming the message you display.
- Reset the counter to zero whenever the user adds something to the cart.
- Change the template to display the counter only if the count is greater than five.

(You'll find hints at http://pragprog.com/wikis/wiki/RailsPlayTime.)