Extracted from:

Agile Web Development with Rails 7

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Agile Web Development with Rails 7

Sam Ruby
with Dave Thomas

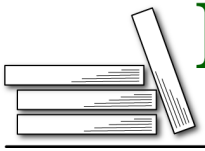
Foreword by
James Duncan Davidson

Edited by Adaobi Obi Tulton

Agile Web Development with Rails 7

Sam Ruby

The Pragmatic Bookshelf
Raleigh, North Carolina



Pragmatic Bookshelf

For our complete catalog of hands-on, practical, and Pragmatic content for software developers, please visit *https://pragprog.com*.

The team that produced this book includes:

CEO: Dave Rankin
COO: Janet Furlow
Managing Editor: Tammy Coron
Development Editor: Adaobi Obi Tulton
Copy Editor: L. Sakhi MacMillan
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics
Founders: Andy Hunt and Dave Thomas

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-929-8
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—May 2023

Iteration C1: Creating the Catalog Listing

We've already created the products controller, used by the seller to administer the Depot application. Now it's time to create a second controller, one that interacts with the paying customers. Let's call it Store:

```
depot> bin/rails generate controller Store index
   create  app/controllers/store_controller.rb
    route  get 'store/index'
   invoke  tailwindcss
   create    app/views/store
   create    app/views/store/index.html.erb
   invoke  test_unit
   create    test/controllers/store_controller_test.rb
   invoke  helper
   create    app/helpers/store_helper.rb
   invoke    test_unit
```

As in the previous chapter, where we used the generate utility to create a controller and associated scaffolding to administer the products, here we've asked it to create a controller (the StoreController class in the store_controller.rb file) containing a single action method, index().

While everything is already set up for this action to be accessed via http://localhost:3000/store/index (feel free to try it!), we can do better. Let's simplify things and make this the root URL for the website. We do this by editing config/routes.rb:

rails7/depot_d/config/routes.rb

```
Rails.application.routes.draw do
  root 'store#index', as: 'store_index'
  resources :products
  # Define your application routes per the DSL in
  # https://guides.rubyonrails.org/routing.html

  # Defines the root path route ("/")
  # root "articles#index"
end
```

We've replaced the get 'store/index' line with a call to define a root path, and in the process we added an as: 'store_index' option. The latter tells Rails to create store_index_path and store_index_url accessor methods, enabling existing code—and tests!—to continue to work correctly. Let's try it. Point a browser at http://localhost:3000/, and up pops our web page. See the following screenshot.

It might not make us rich, but at least we know everything is wired together correctly. It even tells us where to find the template file that draws this page.

Let's start by displaying a list of all the products in our database. We know that eventually we'll have to be more sophisticated, breaking them into categories, but this'll get us going.

We need to get the list of products out of the database and make it available to the code in the view that'll display the table. This means we have to change the index() method in store_controller.rb. We want to program at a decent level of abstraction, so let's assume we can ask the model for a list of the products:

rails7/depot_d/app/controllers/store_controller.rb

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

We asked our customer if she had a preference regarding the order things should be listed in, and we jointly decided to see what happens if we display the products in alphabetical order. We do this by adding an order(:title) call to the Product model.

Now we need to write our view template. To do this, edit the index.html.erb file in app/views/store. (Remember that the path name to the view is built from the name of the controller [store] and the name of the action [index]. The .html.erb part signifies an ERB template that produces an HTML result.)

rails7/depot_d/app/views/store/index.html.erb

```
<div class="w-full">
<% if notice.present? %>
  <p class="py-2 px-3 bg-green-50 mb-5 text-green-500 font-medium rounded-lg
            inline-block" id="notice">
    <%= notice %>
  </p>
<% end %>

<h1 class="font-bold text-xl mb-6 pb-2 border-b-2">
  Your Pragmatic Catalog
</h1>

<ul>
  <% @products.each do |product| %>
    <li class='flex mb-6'>
```

```
      <%= image_tag(product.image_url,
        class: 'object-contain w-40 h-48 shadow mr-6') %>

      <div>
        <h2 class="font-bold text-lg mb-3"><%= product.title %></h2>

        <p>
          <%= sanitize(product.description) %>
        </p>

        <div class="mt-3">
          <%= product.price %>
        </div>
      </div>
    </li>
  <% end %>
</ul>
</div>
```

Note the use of the sanitize() method for the description. This allows us to safely[1] add HTML stylings to make the descriptions more interesting for our customers.

We also used the image_tag() helper method. This generates an HTML <img> tag using its argument as the image source.

A page refresh brings up the display shown in the following screenshot. It's still pretty basic, and it seems to be missing something. The customer happens to be walking by as we ponder this, and she points out that she'd also like to see a decent-looking banner and sidebar on public-facing pages.

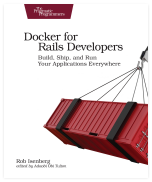
1. https://owasp.org/www-community/attacks/xss/

Your Pragmatic Catalog

Design and Build Great Web APIs

Robust, Reliable, and Resilient APIs are transforming the business world at an increasing pace. Gain the essential skills needed to quickly design, build, and deploy quality web APIs that are robust, reliable, and resilient. Go from initial design through prototyping and implementation to deployment of mission-critical APIs for your organization. Test, secure, and deploy your API with confidence and avoid the "release into production" panic. Tackle just about any API challenge with more than a dozen open-source utilities and common programming patterns you can apply right away.

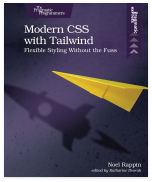
24.95



Docker for Rails Developers

Build, Ship, and Run Your Applications Everywhere Docker does for DevOps what Rails did for web development—it gives you a new set of superpowers. Gone are "works on my machine" woes and lengthy setup tasks, replaced instead by a simple, consistent, Docker-based development environment that will have your team up and running in seconds. Gain hands-on, real-world experience with a tool that's rapidly becoming fundamental to software development. Go from zero all the way to production as Docker transforms the massive leap of deploying your app in the cloud into a baby step.

19.95



Modern CSS with Tailwind

Flexible Styling Without the Fuss Tailwind CSS is an exciting new CSS framework that allows you to design your site by composing simple utility classes to create complex effects. With Tailwind, you can style your text, move your items on the page, design complex page layouts, and adapt your design for devices from a phone to a wide-screen monitor. With this book, you'll learn how to use the Tailwind for its flexibility and its consistency, from the smallest detail of your typography to the entire design of your site.

18.95

At this point in the real world, we'd probably want to call in the design folks. But Pragmatic Web Designer is off getting inspiration on a beach somewhere and won't be back until later in the year, so let's put a placeholder in for now. It's time for another iteration.