Web Engineering:
Task: Catalog display

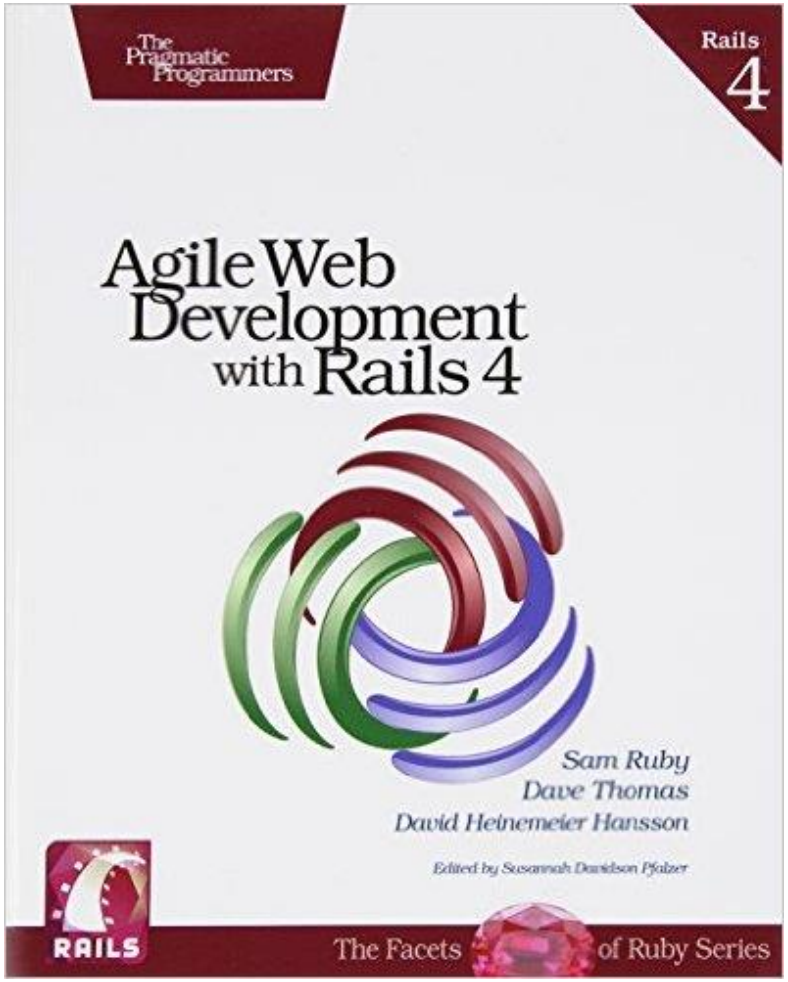
The University of Aizu
Quarter 2, AY 2018

Outline

- Catalog display
  - Creation of the catalog listing
  - Adding a page layout
  - Using a helper to format the price
  - Functional testing and controllers

Literature

- Agile Web Development with Rails 4 (1st edition) by Sam Ruby, Dave Thomas and Devid Hansson, The Pragmatic Bookshelf, 2013.
  - Chapters 8.



Creation of catalog listing

- Our first controller is used by a seller to administer the Depot application.
- Our second controller will interact with paying customers.
  - The name of this controller is *store*.

  depot>rails generate controller Store index
  - The *generate* utility creates a controller: class *StoreController* in the file *store_controller.rb*. This class contains a single action method, *index*.

Creation of catalog listing

- To access this action, type:
  - http://localhost:3000/store/index
- To make it the root URL for the Web site, we edit config/routes.rb :

Download rails40/depot_d/config/routes.rb

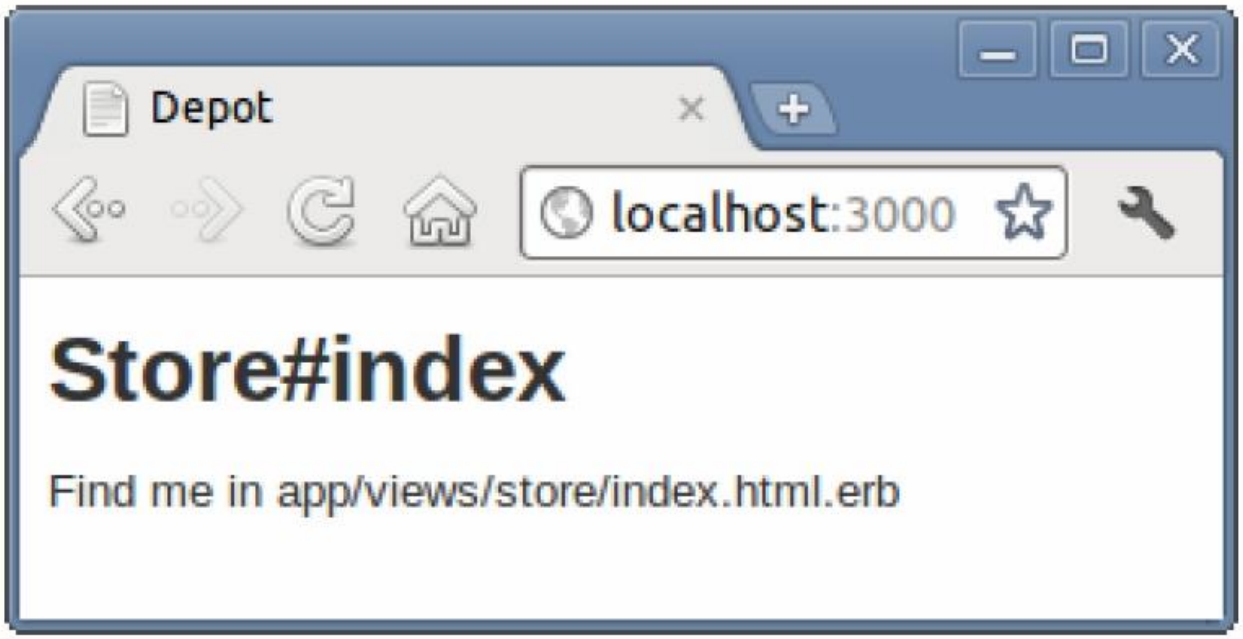
```
Depot::Application.routes.draw do
  get "store/index"
  resources :products

  # The priority is based upon order of creation:
  # first created -> highest priority.
  # See how all your routes lay out with "rake routes".

  # You can have the root of your site routed with "root"
➤ root 'store#index', as: 'store'
  # ...
end
```

Creation of catalog listing

- See the previous slide.
  - At the top of the file, there are the lines added to support the *store* and *products* controllers.
  - You see a commented-out line that defines a root for the website. We have changed on that line the name of the controller (from *welcome* to *store*) and added *as: 'store'*.

Creation of catalog listing

- If you type in the browser:
  http://localhost:3000



- The page event tells us where to find the template file that draws this page.

Creation of catalog listing

- We will start by displaying a simple list of all products in the database.
- We need to get the list of products out of database and make it available to the code in the view that will display the table.
  - We need to change the *index* method in the *store_controller.rb*

Download rails40/depot_d/app/controllers/store_controller.rb

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

Creation of catalog listing

- Now, we need to write our view template. We should edit the file *index.html.erb* in *app/view/store*. See the next slide.

Creation of catalog listing

Download rails40/depot_d/app/views/store/index.html.erb

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

<h1>Your Pragmatic Catalog</h1>

<% @products.each do |product| %>
  <div class="entry">
    <%= image_tag(product.image_url) %>
    <h3><%= product.title %></h3>
    <%= sanitize(product.description) %>
    <div class="price_line">
      <span class="price"><%= product.price %></span>
    </div>
  </div>
<% end %>
```

Comments on the previous slide

- The *sanitize* method allows us to add HTML stylings.
- The *image_tag* method generates an HTML <img> tag using its argument.

Adding a style sheet

Download rails40/depot_d/app/assets/stylesheets/store.css.scss

```
// Place all the styles related to the Store controller here.
// They will automatically be included in application.css.
// You can use Sass (SCSS) here: http://sass-lang.com/

.store {
  h1 {
    margin: 0;
    padding-bottom: 0.5em;
    font:  150% sans-serif;
    color: #226;
    border-bottom: 3px dotted #77d;
  }

  /* An entry in the store catalog */
  .entry {
```

```
    overflow: auto;
    margin-top: 1em;
    border-bottom: 1px dotted #77d;
    min-height: 100px;

    img {
      width: 80px;
      margin-right: 5px;
      margin-bottom: 5px;
      position: absolute;
    }

    h3 {
      font-size: 120%;
      font-family: sans-serif;
      margin-left: 100px;
      margin-top: 0;
      margin-bottom: 2px;
      color: #227;
    }

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

    .price {
      color: #44a;
      font-weight: bold;
      margin-right: 3em;
    }
  }
}
```

Adding a style sheet (cont)

- When you click Refresh button on your browser, you will see the Catalog page, which is ugly.

Adding a page layout

- The name of our layout is *application.html.erb*
- By using only one layout, we can change the look and feel of the entire site by editing just one file in *apps/views/layouts* directory.

Let's update this file to define a banner and a sidebar.

Download rails40/depot_e/app/views/layouts/application.html.erb

```
Line 1 <!DOCTYPE html>
  - <html>
  - <head>
  -   <title>Pragprog Books Online Store</title>
  5   <%= stylesheet_link_tag    "application", media: "all",
  -     "data-turbolinks-track" => true %>
  -   <%= javascript_include_tag "application", "data-turbolinks-track" => true %>
  -   <%= csrf_meta_tags %>
  - </head>
```

- The file continues on the next slide

Adding a page layout

```
<body class="<%= controller.controller_name %>">
  <div id="banner">
    <%= image_tag("logo.png") %>
    <%= @page_title || "Pragmatic Bookshelf" %>
  </div>
  <div id="columns">
    <div id="side">
      <ul>
        <li><a href="http://www....">Home</a></li>
        <li><a href="http://www..../faq">Questions</a></li>
        <li><a href="http://www..../news">News</a></li>
        <li><a href="http://www..../contact">Contact</a></li>
      </ul>
    </div>
    <div id="main">
      <%= yield %>
    </div>
  </div>
</body>
</html>
```

Comments on the previous slide

- Line 5 uses a Rails *stylesheet_link_tag()* helper method to generate a <link> tag to our application's stylesheet and specifies an option to enable turbolinks which transparently works behind the scenes to speed up page changes within your application.
- Line 7 generates a <link> to our application's scripts.
- Line 8: sets-up all the behind the scenes data needed to prevent cross-site request forgery attacks, which will be important once add forms.

Comments on the previous slide

- Line 13: sets the page heading to the value in the instance variable *@page_title*.
- Line 25: Involving *yield*, Rails automatically substitutes in page-specific content: the stuff generated by the view invoked by this request. Here, this will be the catalog page generated by *index.html.erb*

Adding a page layout

- To make this all work, we need
  - rename the file *application.css* to *application.css.scss*.
  - to add the following to this file:

Download rails40/depot_e/app/assets/stylesheets/application.css.scss

```
/*
 * This is a manifest file that'll be compiled into application.css, which will
 * include all the files listed below.
 *
 * Any CSS and SCSS file within this directory, lib/assets/stylesheets,
 * vendor/assets/stylesheets, or vendor/assets/stylesheets of plugins, if any,
 * can be referenced here using a relative path.
 *
 * You're free to add application-wide styles to this file and they'll appear
 * at the top of the compiled file, but it's generally better to create a new
 * file per style scope.
 *
 *= require_self
 *= require_tree .
 */
```

See the next slide

Adding a page layout

```
#banner {
  background: #9c9;
  padding: 10px;
  border-bottom: 2px solid;
  font: small-caps 40px/40px "Times New Roman", serif;
  color: #282;
  text-align: center;

  img {
    float: left;
  }
}

#notice {
  color: #000 !important;
  border: 2px solid red;
  padding: 1em;
  margin-bottom: 2em;
  background-color: #f0f0f0;
  font: bold smaller sans-serif;
}

#columns {
  background: #141;

  #main {
    margin-left: 17em;
    padding: 1em;
    background: white;
  }

  #side {
    float: left;
    padding: 1em 2em;
    width: 13em;
    background: #141;

    ul {
      padding: 0;
      li {
        list-style: none;
        a {
          color: #bfb;
          font-size: small;
        }
      }
    }
  }
}
```

Adding a page layout

- See the previous slide.
- Hit Refresh button on your browser, and you will se the screen similar to this one:

Catalog with layout added

- There is a small problem with how prices are displayed:
  - Instead of 12,34 , it should be $12.34
  - Instead of 13, it should be $13.00

Using the Helper to format the price

- A *number-to-currency* helper method to format prices.
- The line below should be changed:

```
<span class="price"><%= product.price %></span>
```

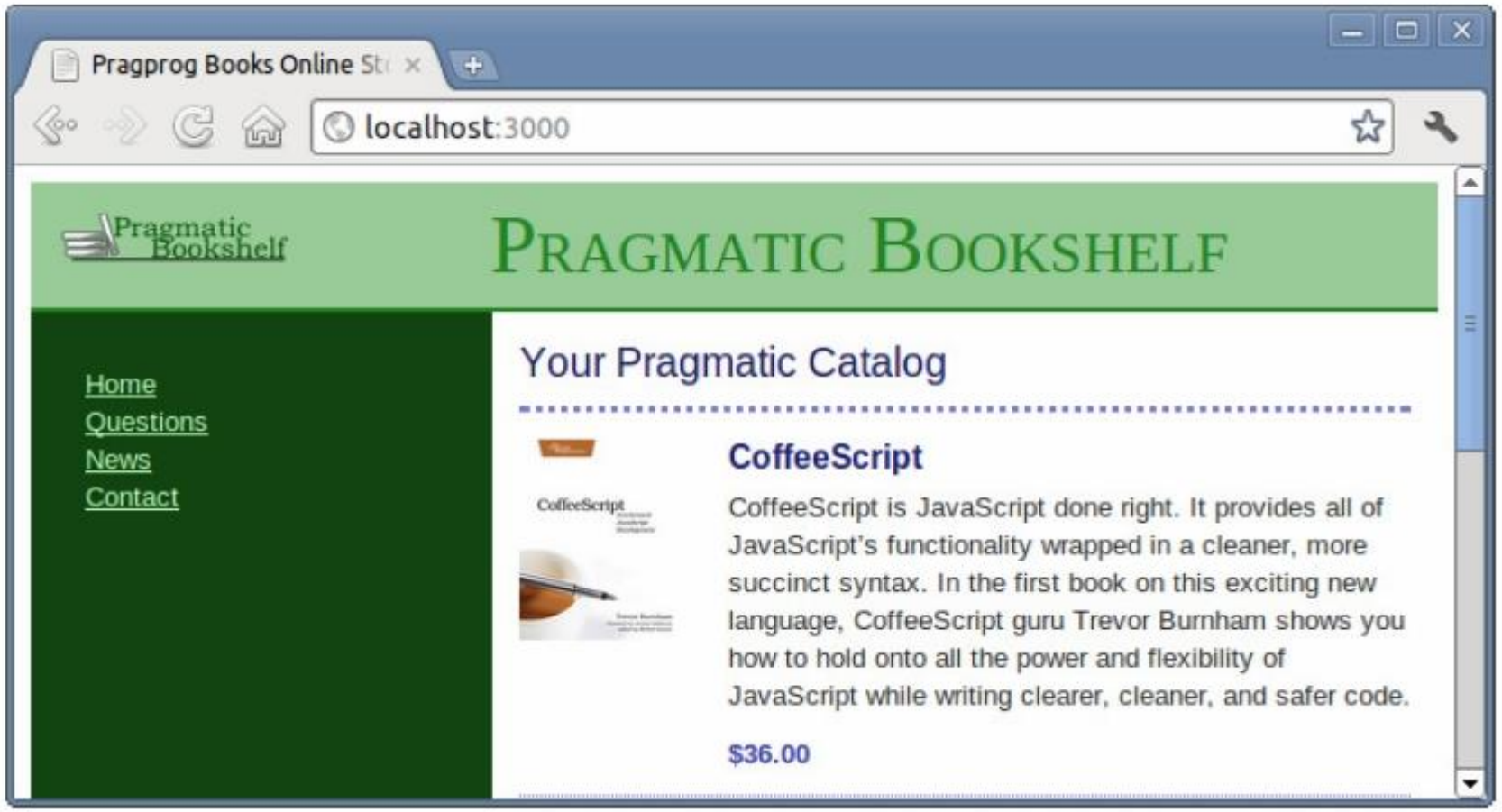
to the following:

Download rails40/depot_e/app/views/store/index.html.erb

```
<span class="price"><%= number_to_currency(product.price) %></span>
```

- Reloading the browser page, we will see the picture (the next slide)
- Now, it is time to test new functionality.

Catalog page



Functional testing of controllers

- Before writing new tests, we have to determine if we have actually broken something:

```
depot>rake test
```

- This time, all is well: nothing was broken.
- We need tests for what we just added.
- Now, we are dealing with the server that processes requests and a user viewing responses in a browser.
- We need the functional tets that verify that the model, view, and controller work well together.

Functional testing of controllers

- Rails generated for us automatically:

Download rails40/depot_d/test/controllers/store_controller_test.rb

```
require 'test_helper'

class StoreControllerTest < ActionController::TestCase
  test "should get index" do
    get :index
    assert_response :success
  end

end
```

- The *should get index* test gets the index and asserts that a successful response is expected.
- We also want to verify that the response contains our layout, our product information, and our number formatting. See the next slide.

Functional testing of controllers

Download rails40/depot_e/test/controllers/store_controller_test.rb

```
require 'test_helper'

class StoreControllerTest < ActionController::TestCase
  test "should get index" do
    get :index
    assert_response :success
    assert_select '#columns #side a', minimum: 4
    assert_select '#main .entry', 3
    assert_select 'h3', 'Programming Ruby 1.9'
    assert_select '.price', /\$[,\d]+\.\d\d/
  end

end
```

- We added FOUR lines:
  - The first select test looks for an element named *a*, that is contained in an element with id with a value of *side*, which is contained within an element with id with value of *columns*.

Comments on the previous slide

- The next three lines that all of our products are displayed.
- The first verifies that there are three elements with class name of entry inside the main portion of the page.
- The next line verifies that there is an h3 element with the title of the Ruby book that we had entered previously.
- The third line verifies that the price is formatted correctly.
- These assertions are based on the test data we put inside our fixture. (See the next slide)

Fixtures

Download rails40/depot_e/test/fixtures/products.yml

```
# Read about fixtures at
# http://api.rubyonrails.org/classes/ActiveRecord/Fixtures.html
one:
  title: MyString
  description: MyText
  image_url: MyString
  price: 9.99

two:
  title: MyString
  description: MyText
  image_url: MyString
  price: 9.99

ruby:
  title:       Programming Ruby 1.9
  description:
    Ruby is the fastest growing and most exciting dynamic
    language out there.  If you need to get working programs
    delivered fast, you should add Ruby to your toolbox.
  price:       49.50
  image_url:   ruby.png
```

Functional testing of controllers

- The type of test that assert_select performs varies base on the type of the second parameter:
  - A number is treated as a quantity
  - A string is treated as an expected result
- A regular expression was used to verify the format of the price: a dollar sign followed by any number (but at least one), commas, or digits; followed by a decimal point; followed by two digits.

Functional testing of controllers

- One important point:
  - Both validation and functional tests will test the behavior of controllers only; they will not retroactively affect any objects that already exist in the database or in fixtures.
- To run functional tests, type:

```
depot> rake test:controllers
```

What we just did

- We put together the basis of the store's catalog display. Our steps were:
  - Create a new controller to handle customer-centric interactions.
  - Implement the default *index* actions.
  - Implement a view (an .html.erb file) and layout to contain it (another .html.erb file)
  - Use a helper to format prices
  - Make use of a CSS stylesheet.
  - Write functional tests for our controller.