5.1 Adding some structure

The *Ruby on Rails Tutorial* is a book on web development, not web design, but it would be depressing to work on an application that looks like *complete* crap, so in this section we'll add some structure to the layout and give it some minimal styling with CSS. In addition to using some custom CSS rules, we'll make use of *Bootstrap*, an open-source web design framework from Twitter.[2] We'll also give our *code* some styling, so to speak, using *partials* to tidy up the layout once it gets a little cluttered.

When building web applications, it is often useful to get a high-level overview of the user interface as early as possible. Throughout the rest of this book, I will thus often include *mockups* (in a web context often called *wireframes*), which are rough sketches of what the eventual application will look like.[3] In this chapter, we will principally be developing the static pages introduced in Section 3.2, including a site logo, a navigation header, and a site footer. A mockup for the most important of these pages, the Home page, appears in Figure 5.1. You can see the final result in Figure 5.9. You'll note that it differs in some details—for example, we'll end up adding a Rails logo on the page—but that's fine, since a mockup need not be exact.

As usual, if you're using Git for version control, now would be a good time to make a new branch:

```
$ git checkout -b filling-in-layout
```

5.1.1 Site navigation

As a first step toward adding links and styles to the sample application, we'll update the site layout file **`application.html.erb`** (last seen in Listing 4.3) with additional HTML structure. This includes some additional divisions, some CSS

[2] Although more recent versions of Bootstrap are now available, this tutorial standardizes on Bootstrap 3 in order to retain compatibility with the design and HTML structure from previous editions.

[3] The mockups in the *Ruby on Rails Tutorial* are made with an excellent online mockup application called Mockingbird.

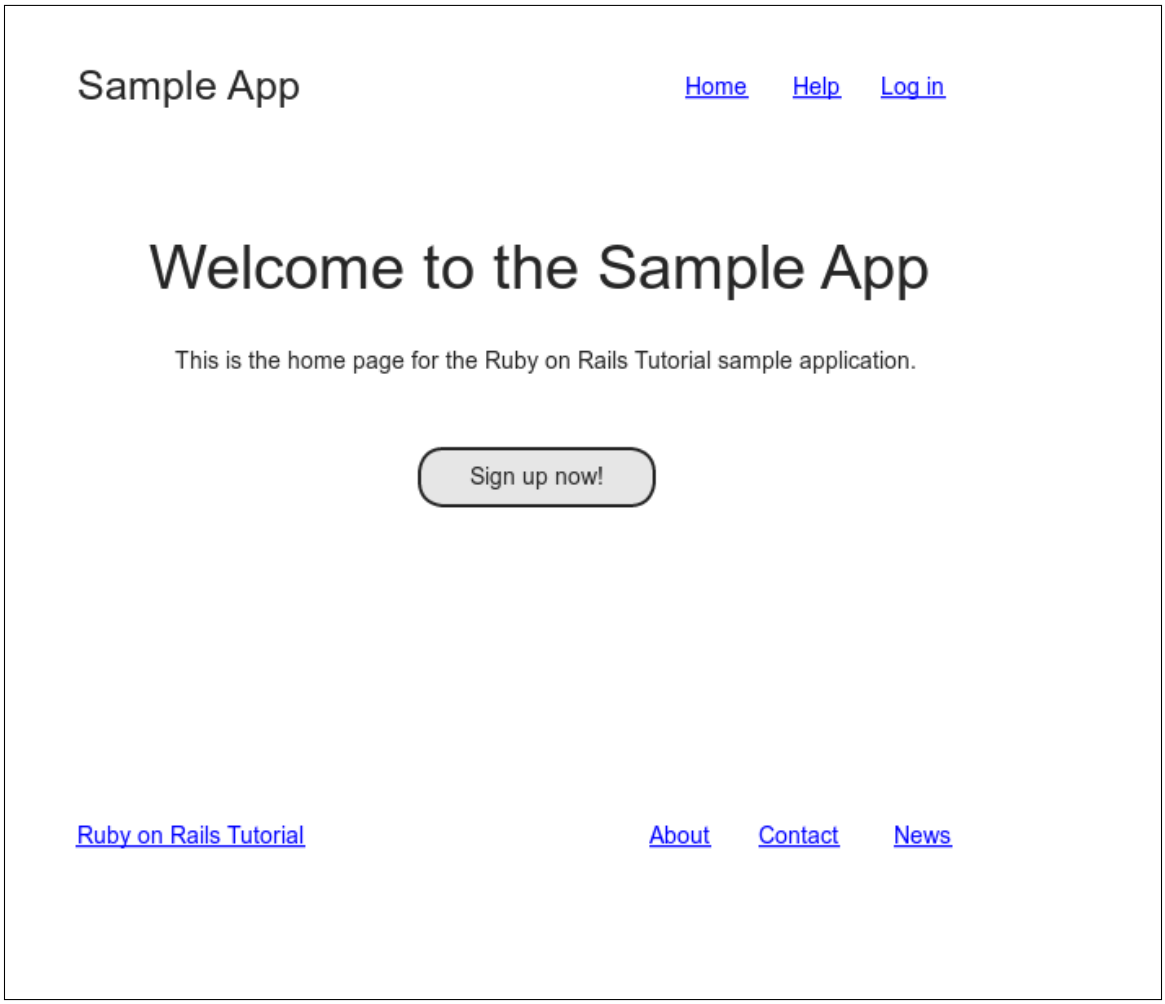

Figure 5.1: A mockup of the sample application's Home page.

classes, and the start of our site navigation. The full file is in Listing 5.1; explanations for the various pieces follow immediately thereafter. If you'd rather not delay gratification, you can see the results in Figure 5.2. (*Note*: it's not (yet) very gratifying.)

Listing 5.1: The site layout with added structure.
app/views/layouts/application.html.erb

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= full_title(yield(:title)) %></title>

    <%= csrf_meta_tags %>
    <%= csp_meta_tag %>

    <%= stylesheet_link_tag 'application', media: 'all',
                                           'data-turbolinks-track': 'reload' %>
    <%= javascript_pack_tag 'application', 'data-turbolinks-track': 'reload' %>
    <!--[if lt IE 9]>
      <script src="//cdnjs.cloudflare.com/ajax/libs/html5shiv/r29/html5.min.js">
      </script>
    <![endif]-->
  </head>
  <body>
    <header class="navbar navbar-fixed-top navbar-inverse">
      <div class="container">
        <%= link_to "sample app", '#', id: "logo" %>
        <nav>
          <ul class="nav navbar-nav navbar-right">
            <li><%= link_to "Home",    '#' %></li>
            <li><%= link_to "Help",    '#' %></li>
            <li><%= link_to "Log in", '#' %></li>
          </ul>
        </nav>
      </div>
    </header>
    <div class="container">
      <%= yield %>
    </div>
  </body>
</html>
```

Let's look at the new elements in Listing 5.1 from top to bottom. As alluded to briefly in Section 3.4.1, Rails uses HTML5 by default (as indicated by the doctype **`<!DOCTYPE html>`**) which at this point most browsers support, but we

can make our site more accessible to older browsers by adding some JavaScript code, known as an "HTML5 shim (or shiv)":[4]

```
<!--[if lt IE 9]>
  <script src="//cdnjs.cloudflare.com/ajax/libs/html5shiv/r29/html5.min.js">
  </script>
<![endif]-->
```

The somewhat odd syntax

```
<!--[if lt IE 9]>
```

includes the enclosed line only if the version of Microsoft Internet Explorer (IE) is less than 9 (`if lt IE 9`). The weird `[if lt IE 9]` syntax is *not* part of Rails; it's actually a conditional comment supported by Internet Explorer browsers for just this sort of situation. It's a good thing, too, because it means we can include the HTML5 shim *only* for IE browsers less than version 9, leaving other browsers such as Firefox, Chrome, and Safari unaffected.

The next section includes a `header` for the site's (plain-text) logo, a couple of divisions (using the `div` tag), and a list of elements with navigation links:

```
<header class="navbar navbar-fixed-top navbar-inverse">
  <div class="container">
    <%= link_to "sample app", '#', id: "logo" %>
    <nav>
      <ul class="nav navbar-nav navbar-right">
        <li><%= link_to "Home",    '#' %></li>
        <li><%= link_to "Help",    '#' %></li>
        <li><%= link_to "Log in", '#' %></li>
      </ul>
    </nav>
  </div>
</header>
```

[4] The words *shim* and *shiv* are used interchangably in this context; the former is the proper term, based on the English word whose meaning is "a washer or thin strip of material used to align parts, make them fit, or reduce wear", while the latter (meaning "a knife or razor used as a weapon") is apparently a play on the name of the shim's original author, Sjoerd Visscher.

Here the **header** tag indicates elements that should go at the top of the page. We've given the **header** tag three *CSS classes*,[5] called **navbar**, **navbar-fixed-top**, and **navbar-inverse**, separated by spaces:

```
<header class="navbar navbar-fixed-top navbar-inverse">
```

All HTML elements can be assigned both classes and *ids*;[6] these are merely labels, and are useful for styling with CSS (Section 5.1.2). The main difference between classes and ids is that classes can be used multiple times on a page, but ids can be used only once. In the present case, all the navbar classes have special meaning to the Bootstrap framework, which we'll install and use in Section 5.1.2.

Inside the **header** tag, we see a **div** tag:

```
<div class="container">
```

The **div** tag is a generic division; it doesn't do anything apart from divide the document into distinct parts. In older-style HTML, **div** tags are used for nearly all site divisions, but HTML5 adds the **header**, **nav**, and **section** elements for divisions common to many applications. In this case, the **div** has a CSS class as well (**container**). As with the **header** tag's classes, this class has special meaning to Bootstrap.

After the div, we encounter some embedded Ruby:

```
<%= link_to "sample app", '#', id: "logo" %>
<nav>
  <ul class="nav navbar-nav navbar-right">
    <li><%= link_to "Home",   '#' %></li>
    <li><%= link_to "Help",   '#' %></li>
    <li><%= link_to "Log in", '#' %></li>
  </ul>
</nav>
```

[5] These are completely unrelated to Ruby classes.

[6] Short for "identification" and pronounced as the separate letters "I D". The usual convention in English is to use all-caps ("ID"), reserving "id" for a term in Freudian psychoanalysis. Because HTML is usually typed in all lower-case letters, though, it's more common in this context to write "id" instead.

This uses the Rails helper `link_to` to create links (which we created directly with the anchor tag `a` in Section 3.2.2); the first argument to `link_to` is the link text, while the second is the URL. We'll fill in the URLs with *named routes* in Section 5.3.3, but for now we use the stub URL `'#'` commonly used in web design (i.e., `'#'` is just a "stub", or placeholder, for the real URL). The third argument is an options hash, in this case adding the CSS id `logo` to the sample app link. (The other three links have no options hash, which is fine since it's optional.) Rails helpers often take options hashes in this way, giving us the flexibility to add arbitrary HTML options without ever leaving Rails.

The second element inside the divs is a list of navigation links, made using the *unordered list* tag `ul`, together with the *list item* tag `li`:

```
<nav>
  <ul class="nav navbar-nav navbar-right">
    <li><%= link_to "Home",    '#' %></li>
    <li><%= link_to "Help",    '#' %></li>
    <li><%= link_to "Log in", '#' %></li>
  </ul>
</nav>
```

The `<nav>` tag, though formally unnecessary here, is used to more clearly communicate the purpose of the navigation links. Meanwhile, the `nav`, `navbar-nav`, and `navbar-right` classes on the `ul` tag have special meaning to Bootstrap and will be styled automatically when we include the Bootstrap CSS in Section 5.1.2. As you can verify by inspecting the navigation in your browser,[7] once Rails has processed the layout and evaluated the embedded Ruby the list looks like this:[8]

```
<nav>
  <ul class="nav navbar-nav navbar-right">
    <li><a href="#">Home</a></li>
    <li><a href="#">Help</a></li>
    <li><a href="#">Log in</a></li>
  </ul>
</nav>
```

[7] All modern browsers have the capability to inspect the HTML source of a page. If you've never used a web inspector before, do a web search for something like "web inspector <name of browser>" to learn more.

[8] The spacing might look slightly different, which is fine because (as noted in Section 3.4.1) HTML is insensitive to whitespace.

This is the text that will be returned to the browser.

The final part of the layout is a **div** for the main content:

```
<div class="container">
  <%= yield %>
</div>
```

As before, the **container** class has special meaning to Bootstrap. As we learned in Section 3.4.3, the **yield** method inserts the contents of each page into the site layout.

Apart from the site footer, which we'll add in Section 5.1.3, our layout is now complete, and we can look at the results by visiting the Home page. To take advantage of the upcoming style elements, we'll add some extra elements to the **home.html.erb** view (Listing 5.2).

Listing 5.2: The Home page with a link to the signup page.
app/views/static_pages/home.html.erb

```
<div class="center jumbotron">
  <h1>Welcome to the Sample App</h1>

  <h2>
    This is the home page for the
    <a href="https://www.railstutorial.org/">Ruby on Rails Tutorial</a>
    sample application.
  </h2>

  <%= link_to "Sign up now!", '#', class: "btn btn-lg btn-primary" %>
</div>

<%= link_to image_tag("rails.svg", alt: "Rails logo", width: "200"),
                      "https://rubyonrails.org/" %>
```

In preparation for adding users to our site in Chapter 7, the first **link_to** creates a stub link of the form

```
<a href="#" class="btn btn-lg btn-primary">Sign up now!</a>
```

In the **div** tag, the **jumbotron** CSS class has a special meaning to Bootstrap, as do the **btn**, **btn-lg**, and **btn-primary** classes in the signup button.

The second **link_to** shows off the **image_tag** helper, which takes as arguments the path to an image and an optional options hash, in this case setting the **alt** and **width** attributes of the image tag using symbols. For this to work, there needs to be an image called **rails.svg**, which you should download from the Learn Enough website at https://cdn.learnenough.com/rails.svg and place in the **app/assets/images/** directory.

If you're using the cloud IDE or another Unix-like system, you can accomplish this with the **curl** utility, as shown in Listing 5.3.[9]

Listing 5.3: Downloading an image.

```
$ curl -o app/assets/images/rails.svg -OL https://cdn.learnenough.com/rails.svg
```

Because we used the **image_tag** helper in Listing 5.2, Rails will automatically find any images in the **app/assets/images/** directory using the asset pipeline (Section 5.2).

Now we're finally ready to see the fruits of our labors. You may have to restart the Rails server to see the changes (Box 1.2), and the results should appear as shown in Figure 5.2.

To make the effects of **image_tag** clearer, let's look at the HTML it produces by inspecting the image in our browser:[10]

```
<img alt="Rails logo" width="200px" src="/assets/rails-<long string>.svg">
```

Here the **<long string>** is a random value added by Rails to ensure that the filename is unique, which causes browsers to load images properly when they have been updated (instead of retrieving them from the browser cache). Note that the **src** attribute *doesn't* include **images**, instead using an **assets** directory common to all assets (images, JavaScript, CSS, etc.). On the server, Rails

[9] See *Learn Enough Command Line to Be Dangerous* for more information about **curl**.

[10] You might notice that the **img** tag, rather than looking like `<img>...</img>`, instead looks like `<img ... />`. Tags that follow this form are known as *self-closing* tags.

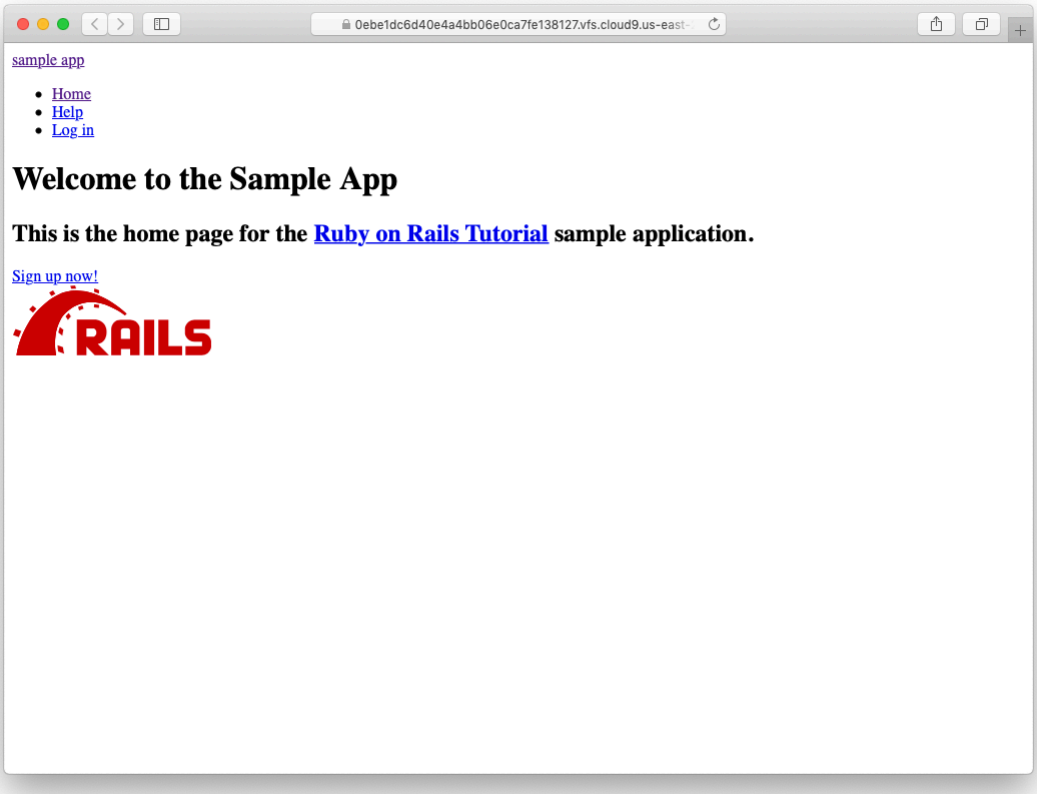

Figure 5.2: The Home page with no custom CSS.

associates images in the `assets` directory with the proper `app/assets/images` directory, but as far as the browser is concerned all the assets look like they are in the same directory, which allows them to be served faster. Meanwhile, the `alt` attribute is what will be displayed if the page is accessed by a program that can't display images (such as screen readers for the visually impaired).

As for the result shown in Figure 5.2, it might look a little underwhelming. Happily, though, we've done a good job of giving our HTML elements sensible classes, which puts us in a great position to add style to the site with CSS.

Exercises

Solutions to the exercises are available to all Rails Tutorial purchasers here.

To see other people's answers and to record your own, subscribe to the Rails Tutorial course or to the Learn Enough All Access Bundle.

1. It's well-known that no web page is complete without a cat image. Using the command in Listing 5.4, arrange to download the kitten pic shown in Figure 5.3.[11]

2. Using the `mv` command, move `kitten.jpg` to the correct asset directory for images (Section 5.2.1).

3. Using `image_tag`, add `kitten.jpg` to the Home page, as shown in Figure 5.4.

Listing 5.4: Downloading a cat picture from the Internet.

```
$ curl -OL https://cdn.learnenough.com/kitten.jpg
```

[11] Image retrieved from https://www.flickr.com/photos/deborah_s_perspective/14144861329 on 2016-01-09. Copyright © 2009 by Deborah and used unaltered under the terms of the Creative Commons Attribution 2.0 Generic license.

Figure 5.3: An obligatory kitten pic.

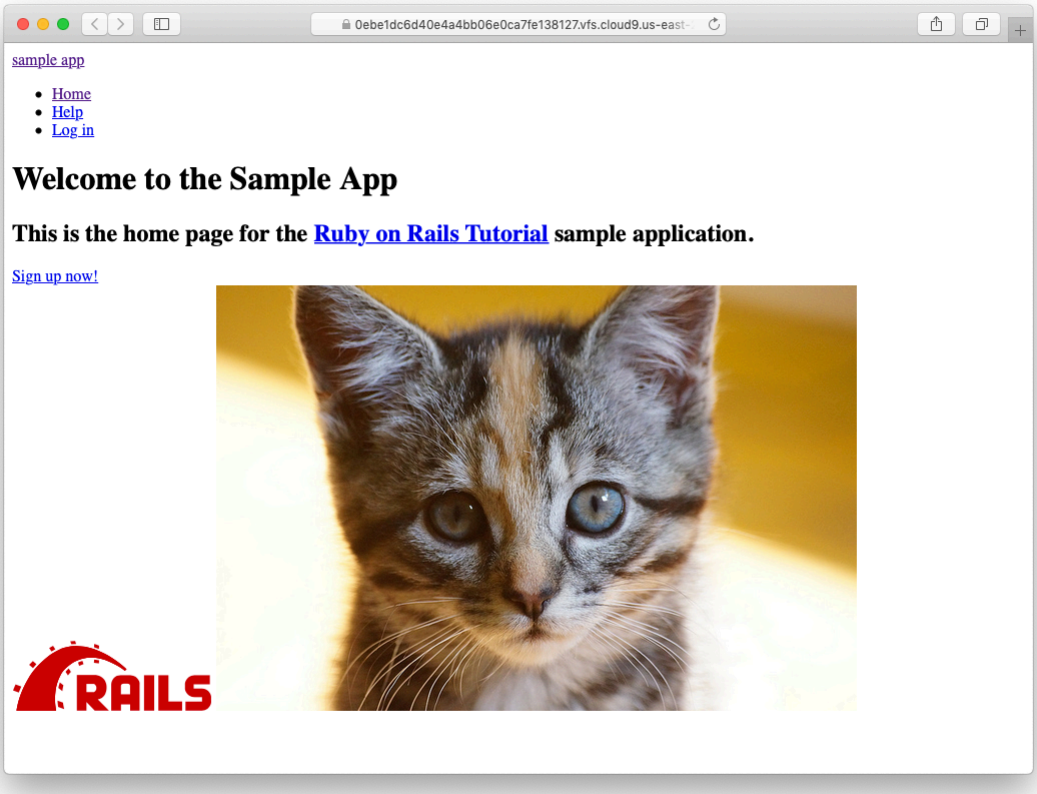

Figure 5.4: The result of adding a kitten image to the Home page.