5.1.2 Bootstrap and custom CSS

In Section 5.1.1, we associated many of the HTML elements with CSS classes, which gives us considerable flexibility in constructing a layout based on CSS. As noted in Section 5.1.1, many of these classes are specific to Bootstrap, a CSS framework that makes it easy to add nice web design and user interface elements to an HTML5 application. In this section, we'll combine Bootstrap with some custom CSS rules to start adding some style to the sample application. It's worth noting that using Bootstrap automatically makes our application's design *responsive*, ensuring that it looks sensible across a wide range of devices.

Our first step is to add Bootstrap, which in Rails applications can be accomplished with the `bootstrap-sass` gem, as shown in Listing 5.5.[12] The Bootstrap framework natively uses the Less CSS language for making dynamic stylesheets, but the Rails asset pipeline supports the (very similar) Sass language by default (Section 5.2), so `bootstrap-sass` converts Less to Sass and makes all the necessary Bootstrap files available to the current application.

Listing 5.5: Adding the `bootstrap-sass` gem to the **`Gemfile`**.

```
source 'https://rubygems.org'

gem 'rails',          '6.0.1'
gem 'bootstrap-sass', '3.4.1'
gem 'puma',           '3.12.1'
.
.
.
```

To install Bootstrap, we run **`bundle install`** as usual:

```
$ bundle install
```

Although **`rails generate`** automatically creates a separate CSS file for each controller, it's surprisingly hard to include them all properly and in the

[12]As always, you should use the version numbers listed at gemfiles-6th-ed.railstutorial.org instead of the ones listed here.

right order, so for simplicity we'll put all of the CSS needed for this tutorial in a single file. The first step toward getting custom CSS to work is to create such a custom CSS file:

```
$ touch app/assets/stylesheets/custom.scss
```

(This uses the **`touch`** trick from Section 3.3.3 en route, but you can create the file however you like.) Here both the directory name and filename extension are important. The directory

```
app/assets/stylesheets/
```

is part of the asset pipeline (Section 5.2), and any stylesheets in this directory will automatically be included as part of the **`application.css`** file included in the site layout. Furthermore, the filename **`custom.scss`** includes the **`.scss`** extension, which indicates a "Sassy CSS" file and arranges for the asset pipeline to process the file using Sass. (We won't be using Sass until Section 5.2.2, but it's needed now for the `bootstrap-sass` gem to work its magic.)

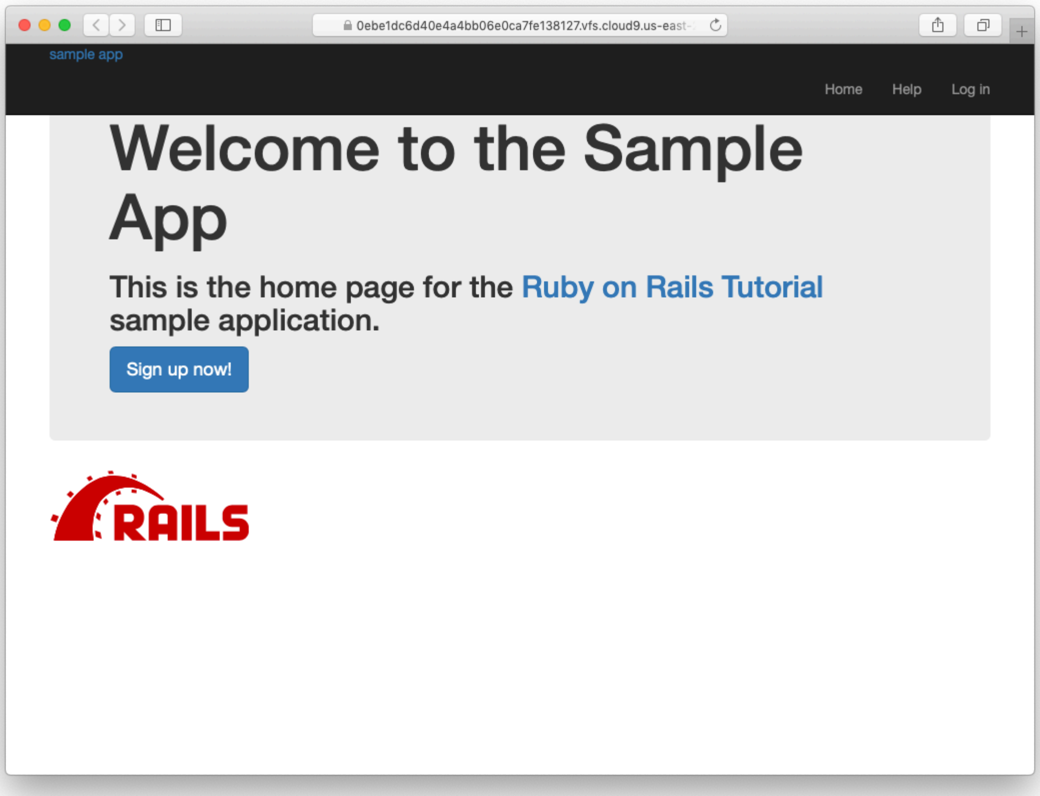
Inside the file for the custom CSS, we can use the **`@import`** function to include Bootstrap (together with the associated Sprockets utility), as shown in Listing 5.6.[13]

Listing 5.6: Adding Bootstrap CSS.
app/assets/stylesheets/custom.scss

```
@import "bootstrap-sprockets";
@import "bootstrap";
```

The two lines in Listing 5.6 include the entire Bootstrap CSS framework. After restarting the webserver to incorporate the changes into the development application (by pressing Ctrl-C and then running **`rails server`** as in Section 1.2.2), the results appear as in Figure 5.5. The placement of the text isn't

[13]If these steps seem mysterious, take heart: I'm just following the instructions from the bootstrap-sass README file.



Welcome to the Sample App

This is the home page for the Ruby on Rails Tutorial sample application.

Figure 5.5: The sample application with Bootstrap CSS.

good and the logo doesn't have any style, but the colors and signup button look promising.

Next we'll add some CSS that will be used site-wide for styling the layout and each individual page, as shown in Listing 5.7. The result is shown in Figure 5.6. (There are quite a few rules in Listing 5.7; to get a sense of what a CSS rule does, it's often helpful to comment it out using CSS comments, i.e., by putting it inside `/* … */`, and seeing what changes.)

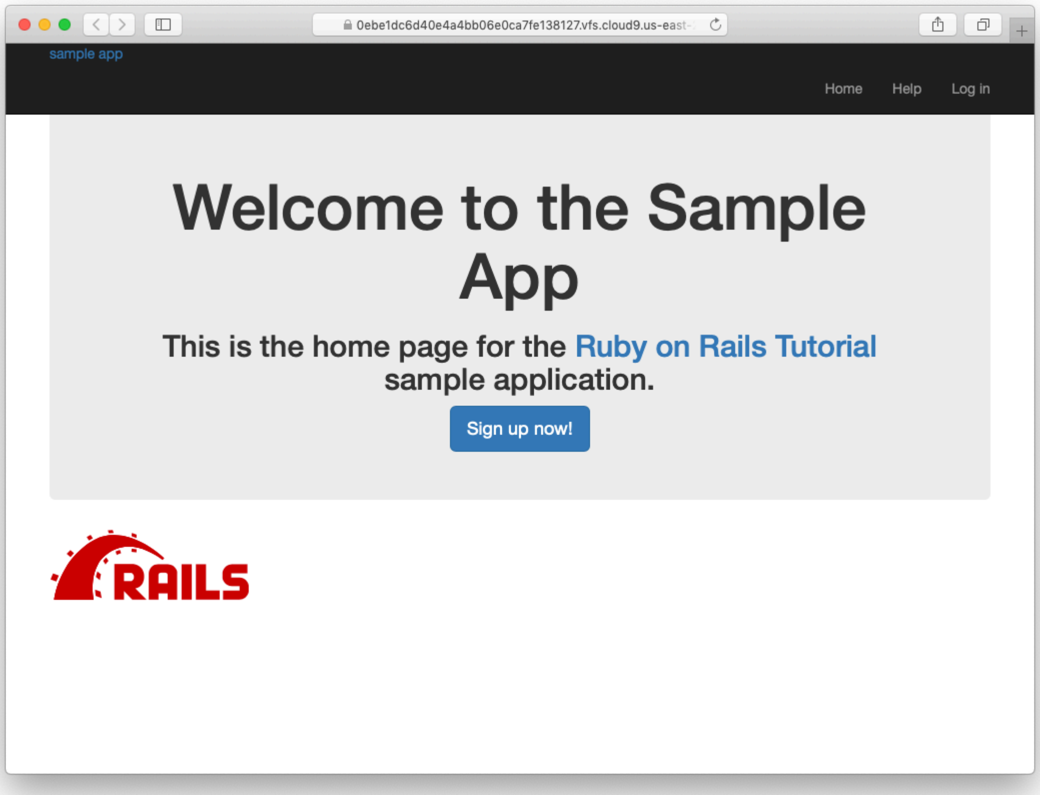
Listing 5.7: Adding CSS for some universal styling applying to all pages.
app/assets/stylesheets/custom.scss

```
@import "bootstrap-sprockets";
@import "bootstrap";

/* universal */

body {
  padding-top: 60px;
}

section {
  overflow: auto;
}

textarea {
  resize: vertical;
}

.center {
  text-align: center;
}

.center h1 {
  margin-bottom: 10px;
}
```

Note that the CSS in Listing 5.7 has a consistent form. In general, CSS rules refer either to a class, an id, an HTML tag, or some combination thereof, followed by a list of styling commands. For example,

```
body {
  padding-top: 60px;
}
```

puts 60 pixels of padding at the top of the page. Because of the **navbar-fixed-top** class in the **header** tag, Bootstrap fixes the navigation bar to the top of the page, so the padding serves to separate the main text from the navigation. (Because the default navbar color changed after Bootstrap 2.0, we need the **navbar-inverse** class to make it dark instead of light.) Meanwhile, the CSS in the rule



Welcome to the Sample App

This is the home page for the Ruby on Rails Tutorial sample application.

Figure 5.6: Adding some spacing and other universal styling.

```
.center {
  text-align: center;
}
```

associates the `center` class with the `text-align: center` property. In other words, the dot `.` in `.center` indicates that the rule styles a class. (As we'll see in Listing 5.9, the pound sign `#` identifies a rule to style a CSS *id*.) This means that elements inside any tag (such as a `div`) with class `center` will be centered on the page. (We saw an example of this class in Listing 5.2.)

Although Bootstrap comes with CSS rules for nice typography, we'll also add some custom rules for the appearance of the text on our site, as shown in Listing 5.8. (Not all of these rules apply to the Home page, but each rule here will be used at some point in the sample application.) The result of Listing 5.8 is shown in Figure 5.7.

Listing 5.8: Adding CSS for nice typography.
app/assets/stylesheets/custom.scss

```
@import "bootstrap-sprockets";
@import "bootstrap";
.
.
.
/* typography */

h1, h2, h3, h4, h5, h6 {
  line-height: 1;
}

h1 {
  font-size: 3em;
  letter-spacing: -2px;
  margin-bottom: 30px;
  text-align: center;
}

h2 {
  font-size: 1.2em;
  letter-spacing: -1px;
  margin-bottom: 30px;
  text-align: center;
  font-weight: normal;
  color: #777;
```

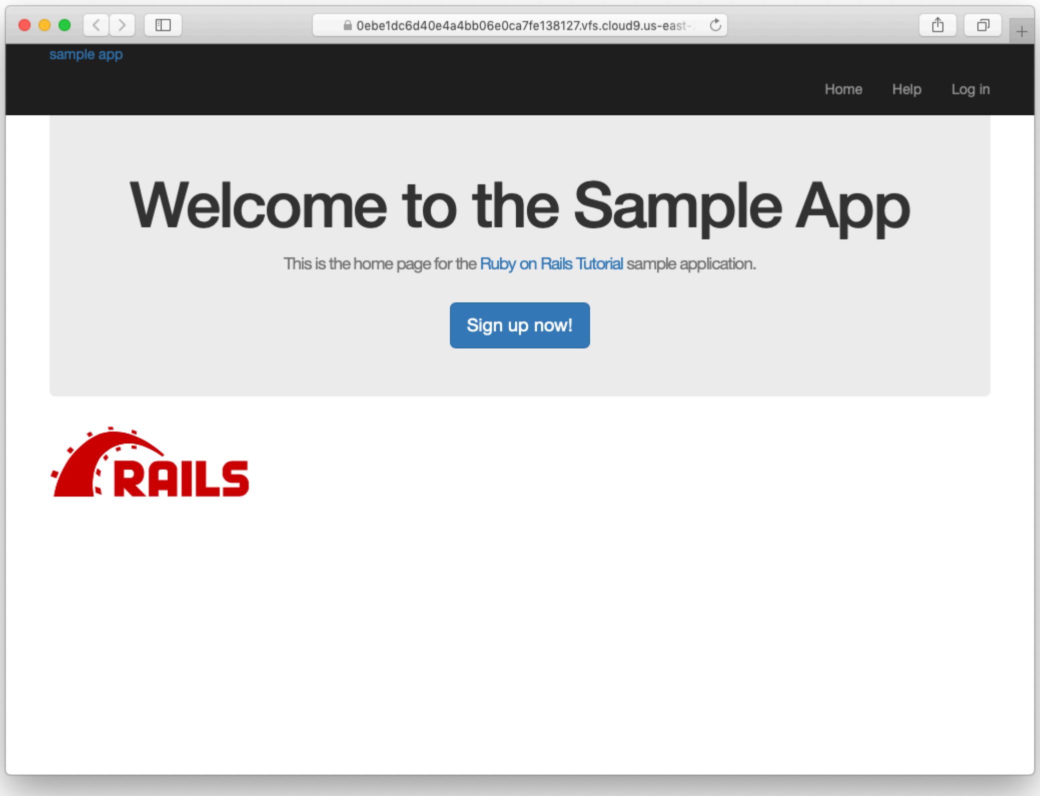

0ebe1dc6d40e4a4bb06e0ca7fe138127.vfs.cloud9.us-east-
sample app
Home
Help
Log in
Welcome to the Sample App
This is the home page for the Ruby on Rails Tutorial sample application.
Sign up now!
RAILS

Figure 5.7: Adding some typographic styling.

```
}

p {
  font-size: 1.1em;
  line-height: 1.7em;
}
```

Finally, we'll add some rules to style the site's logo, which simply consists of the text "sample app". The CSS in Listing 5.9 converts the text to uppercase and modifies its size, color, and placement. (We've used a CSS id because we expect the site logo to appear on the page only once, but you could use a class instead.)

Listing 5.9: Adding CSS for the site logo.
app/assets/stylesheets/custom.scss

```
@import "bootstrap-sprockets";
@import "bootstrap";
.
.
.
/* header */

#logo {
  float: left;
  margin-right: 10px;
  font-size: 1.7em;
  color: #fff;
  text-transform: uppercase;
  letter-spacing: -1px;
  padding-top: 9px;
  font-weight: bold;
}

#logo:hover {
  color: #fff;
  text-decoration: none;
}
```

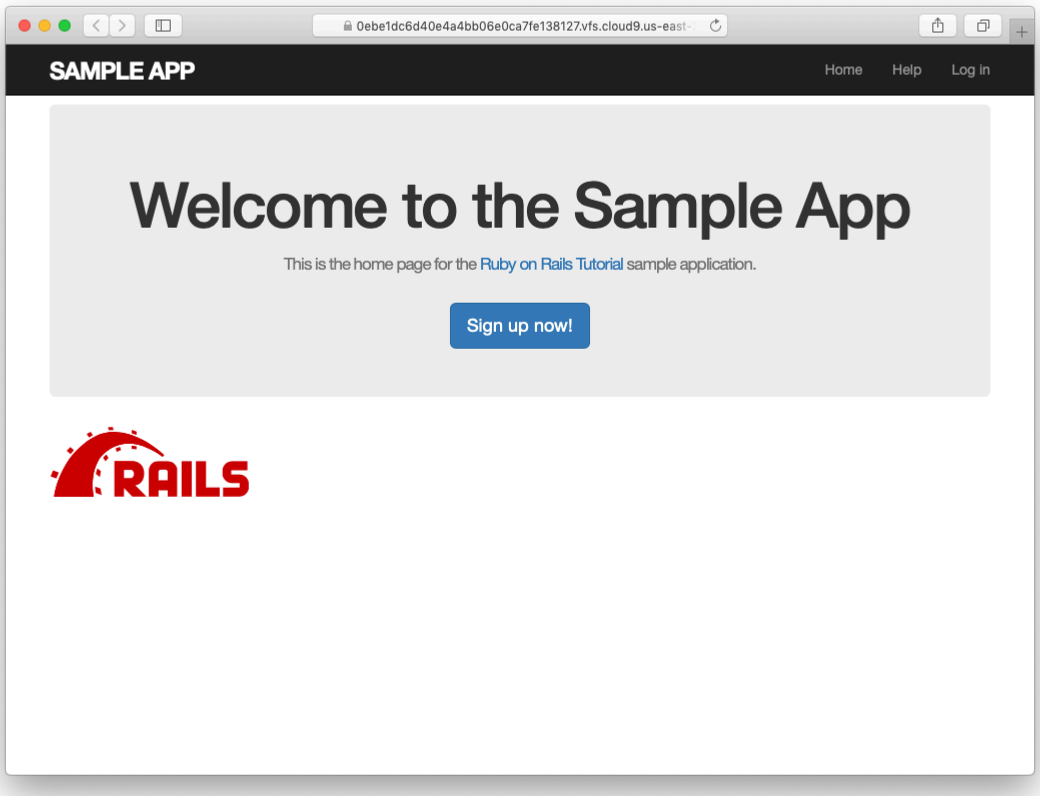
Here `color: #fff` changes the color of the logo to white. HTML colors can be coded with three pairs of base-16 (hexadecimal) numbers, one each for the primary colors red, green, and blue (in that order). The code `#ffffff` maxes out all three colors, yielding pure white, and `#fff` is a shorthand for the full `#ffffff`. The CSS standard also defines a large number of synonyms for common HTML colors, including `white` for `#fff`. The result of the CSS in Listing 5.9 is shown in Figure 5.8.

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. Using code like that shown in Listing 5.10, comment out the cat image



Welcome to the Sample App

This is the home page for the Ruby on Rails Tutorial sample application.

Figure 5.8: The sample app with nicely styled logo.

from Section 5.1.1. Verify using a web inspector that the HTML for the image no longer appears in the page source.

2. By adding the CSS in Listing 5.11 to **custom.scss**, hide all images in the application—currently just the Rails logo on the Home page). Verify with a web inspector that, although the image doesn't appear, the HTML source is still present.

Listing 5.10: Code to comment out embedded Ruby.

```
<%#= image_tag("kitten.jpg", alt: "Kitten") %>
```

Listing 5.11: CSS to hide all images.

```
img {
  display: none;
}
```

5.1.3 Partials

Although the layout in Listing 5.1 serves its purpose, it's getting a little cluttered. The HTML shim takes up three lines and uses weird IE-specific syntax, so it would be nice to tuck it away somewhere on its own. In addition, the header HTML forms a logical unit, so it should all be packaged up in one place. The way to achieve this in Rails is to use a facility called *partials*. Let's first take a look at what the layout looks like after the partials are defined (Listing 5.12).

Listing 5.12: The site layout with partials for the stylesheets and header.
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' %>
    <%= render 'layouts/shim' %>
  </head>
  <body>
    <%= render 'layouts/header' %>
    <div class="container">
      <%= yield %>
    </div>
  </body>
</html>
```

In Listing 5.12, we've replaced the HTML shim stylesheet lines with a single call to a Rails helper called **`render`**:

```
<%= render 'layouts/shim' %>
```

The effect of this line is to look for a file called **`app/views/layouts/_shim.html.erb`**, evaluate its contents, and insert the results into the view.[14] (Recall that **`<%= ... %>`** is the embedded Ruby syntax needed to evaluate a Ruby expression and then insert the results into the template.) Note the leading underscore on the filename **`_shim.html.erb`**; this underscore is the universal convention for naming partials, and among other things makes it possible to identify all the partials in a directory at a glance.

To get the partial to work, we have to create the corresponding file and fill it with some content. In the case of the shim partial, this is just the three lines of shim code from Listing 5.1. The result appears in Listing 5.13.

Listing 5.13: A partial for the HTML shim.
`app/views/layouts/_shim.html.erb`

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

[14]Many Rails developers use a **`shared`** directory for partials shared across different views. I prefer to use the **`shared`** folder for utility partials that are useful on multiple views, while putting partials that are literally on every page (as part of the site layout) in the **`layouts`** directory. (We'll create the **`shared`** directory starting in Chapter 7.) That seems to me a logical division, but putting them all in the **`shared`** folder certainly works fine, too.

Similarly, we can move the header material into the partial shown in Listing 5.14 and insert it into the layout with another call to **`render`**. (As usual with partials, you will have to create the file by hand using your text editor.)

Listing 5.14: A partial for the site header.
app/views/layouts/_header.html.erb

```
<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>
```

Now that we know how to make partials, let's add a site footer to go along with the header. By now you can probably guess that we'll call it **`_footer.-html.erb`** and put it in the layouts directory (Listing 5.15).[15]

Listing 5.15: A partial for the site footer.
app/views/layouts/_footer.html.erb

```
<footer class="footer">
  <small>
    The <a href="https://www.railstutorial.org/">Ruby on Rails Tutorial</a>
    by <a href="https://www.michaelhartl.com/">Michael Hartl</a>
  </small>
  <nav>
    <ul>
      <li><%= link_to "About",   '#' %></li>
      <li><%= link_to "Contact", '#' %></li>
      <li><a href="https://news.railstutorial.org/">News</a></li>
    </ul>
  </nav>
</footer>
```

[15]You may wonder why we use both the **`footer`** tag and **`.footer`** class. The answer is that the tag has a clear meaning to human readers, and the class is used by Bootstrap. Using a **`div`** tag in place of **`footer`** would work as well.

As with the header, in the footer we've used **`link_to`** for the internal links to the About and Contact pages and stubbed out the URLs with `'#'` for now. (As with **`header`**, the **`footer`** tag is new in HTML5.)

We can render the footer partial in the layout by following the same pattern as the stylesheets and header partials (Listing 5.16).

Listing 5.16: The site layout with a footer partial.
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' %>
    <%= render 'layouts/shim' %>
  </head>
  <body>
    <%= render 'layouts/header' %>
    <div class="container">
      <%= yield %>
      <%= render 'layouts/footer' %>
    </div>
  </body>
</html>
```

Next, we'll add some styling for the footer, as shown in Listing 5.17. The results appear in Figure 5.9.

Listing 5.17: Adding the CSS for the site footer.
app/assets/stylesheets/custom.scss

```
.
.
.
/* footer */

footer {
  margin-top: 45px;
  padding-top: 5px;
  border-top: 1px solid #eaeaea;
  color: #777;
}

footer a {
  color: #555;
}

footer a:hover {
  color: #222;
}

footer small {
  float: left;
}

footer ul {
  float: right;
  list-style: none;
}

footer ul li {
  float: left;
  margin-left: 15px;
}
```

Exercises

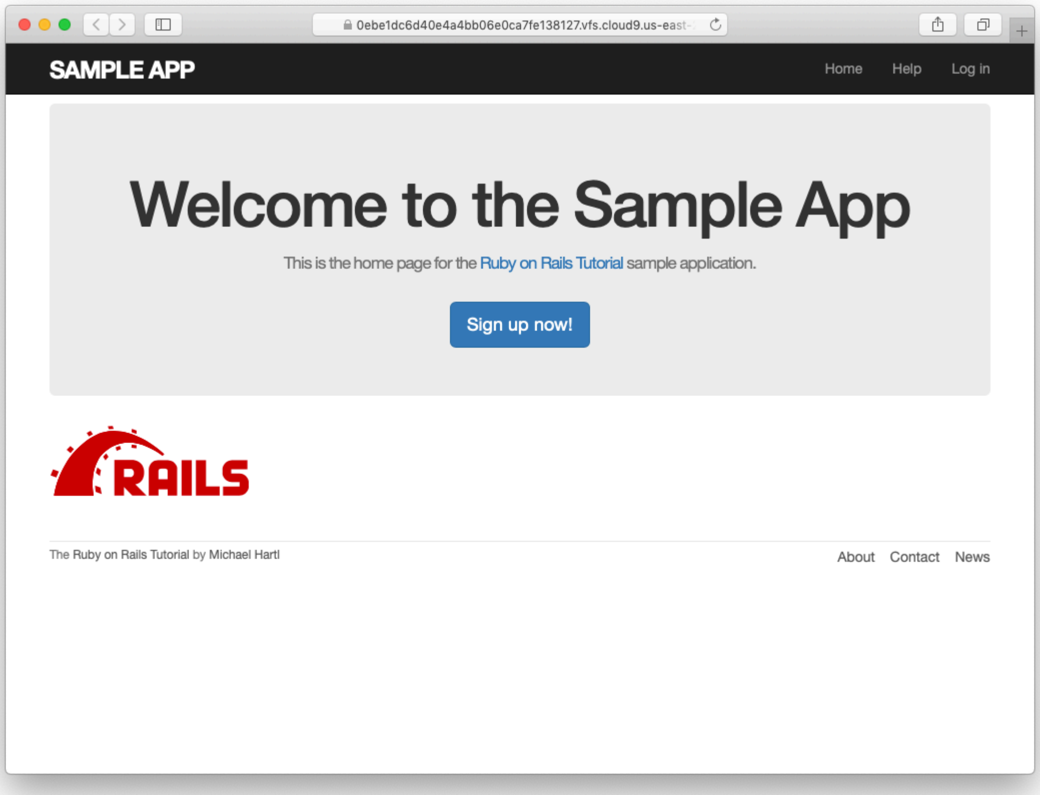
Solutions to the exercises are available to all Rails Tutorial purchasers here.

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. Replace the default Rails head with the call to **`render`** shown in Listing 5.18. *Hint*: For convenience, cut the default header rather than just deleting it.

2. Because we haven't yet created the partial needed by Listing 5.18, the tests should be RED. Confirm that this is the case.

3. Create the necessary partial in the **`layouts`** directory, paste in the contents, and verify that the tests are now GREEN again.



Welcome to the Sample App

This is the home page for the Ruby on Rails Tutorial sample application.

The Ruby on Rails Tutorial by Michael Hartl

Figure 5.9: The Home page with an added footer.