Page	**URL**	**Base title**	**Variable title**
Home	/static_pages/home	`"Ruby on Rails Tutorial Sample App"`	`"Home"`
Help	/static_pages/help	`"Ruby on Rails Tutorial Sample App"`	`"Help"`
About	/static_pages/about	`"Ruby on Rails Tutorial Sample App"`	`"About"`

Table 3.2: The (mostly) static pages for the sample app.

3.4 Slightly dynamic pages

Now that we've created the actions and views for some static pages, we'll make them *slightly* dynamic by adding some content that changes on a per-page basis: we'll have the title of each page change to reflect its content. Whether a changing title represents *truly* dynamic content is debatable, but in any case it lays the necessary foundation for unambiguously dynamic content in Chapter 7.

Our plan is to edit the Home, Help, and About pages to make page titles that change on each page. This will involve using the `<title>` tag in our page views. Most browsers display the contents of the title tag at the top of the browser window, and it is also important for search-engine optimization. We'll be using the full "Red, Green, Refactor" cycle: first by adding simple tests for our page titles (RED), then by adding titles to each of our three pages (GREEN), and finally using a *layout* file to eliminate duplication (Refactor). By the end of this section, all three of our static pages will have titles of the form "<page name> | Ruby on Rails Tutorial Sample App", where the first part of the title will vary depending on the page (Table 3.2).

The `rails new` command (Listing 3.1) creates a layout file by default, but it's instructive to ignore it initially, which we can do by changing its name:

```
$ mv app/views/layouts/application.html.erb layout_file   # temporary change
```

You wouldn't normally do this in a real application, but it's easier to understand the purpose of the layout file if we start by disabling it.

3.4.1 Testing titles (Red)

To add page titles, we need to learn (or review) the structure of a typical web page, which takes the form shown in Listing 3.25. (This is covered in much more depth in *Learn Enough HTML to Be Dangerous*.)

Listing 3.25: The HTML structure of a typical web page.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Greeting</title>
  </head>
  <body>
    <p>Hello, world!</p>
  </body>
</html>
```

The structure in Listing 3.25 includes a *document type*, or doctype, declaration at the top to tell browsers which version of HTML we're using (in this case, HTML5);[10] a **`head`** section, in this case with "Greeting" inside a **`title`** tag; and a **`body`** section, in this case with "Hello, world!" inside a **`p`** (paragraph) tag. (The indentation is optional—HTML is not sensitive to whitespace, and ignores both tabs and spaces—but it makes the document's structure easier to see.)

We'll write simple tests for each of the titles in Table 3.2 by combining the tests in Listing 3.17 with the **`assert_select`** method, which lets us test for the presence of a particular HTML tag (sometimes called a "selector", hence the name):[11]

```
assert_select "title", "Home | Ruby on Rails Tutorial Sample App"
```

In particular, the code above checks for the presence of a **`<title>`** tag containing the string "Home | Ruby on Rails Tutorial Sample App". Applying this

[10] HTML changes with time; by explicitly making a doctype declaration we make it likelier that browsers will render our pages properly in the future. The simple doctype **`<!DOCTYPE html>`** is characteristic of the latest HTML standard, HTML5.

[11] For a list of common minitest assertions, see the table of available assertions in the Rails Guides testing article.

idea to all three static pages gives the tests shown in Listing 3.26.

Listing 3.26: The Static Pages controller test with title tests. RED
test/controllers/static_pages_controller_test.rb

```
require 'test_helper'

class StaticPagesControllerTest < ActionDispatch::IntegrationTest

  test "should get home" do
    get static_pages_home_url
    assert_response :success
    assert_select "title", "Home | Ruby on Rails Tutorial Sample App"
  end

  test "should get help" do
    get static_pages_help_url
    assert_response :success
    assert_select "title", "Help | Ruby on Rails Tutorial Sample App"
  end

  test "should get about" do
    get static_pages_about_url
    assert_response :success
    assert_select "title", "About | Ruby on Rails Tutorial Sample App"
  end
end
```

With the tests from Listing 3.26 in place, you should verify that the test suite is currently RED:

Listing 3.27: RED

```
$ rails test
3 tests, 6 assertions, 3 failures, 0 errors, 0 skips
```

3.4.2 Adding page titles (Green)

Now we'll add a title to each page, getting the tests from Section 3.4.1 to pass in the process. Applying the basic HTML structure from Listing 3.25 to the custom Home page from Listing 3.12 yields Listing 3.28.

Listing 3.28: The view for the Home page with full HTML structure. RED
app/views/static_pages/home.html.erb

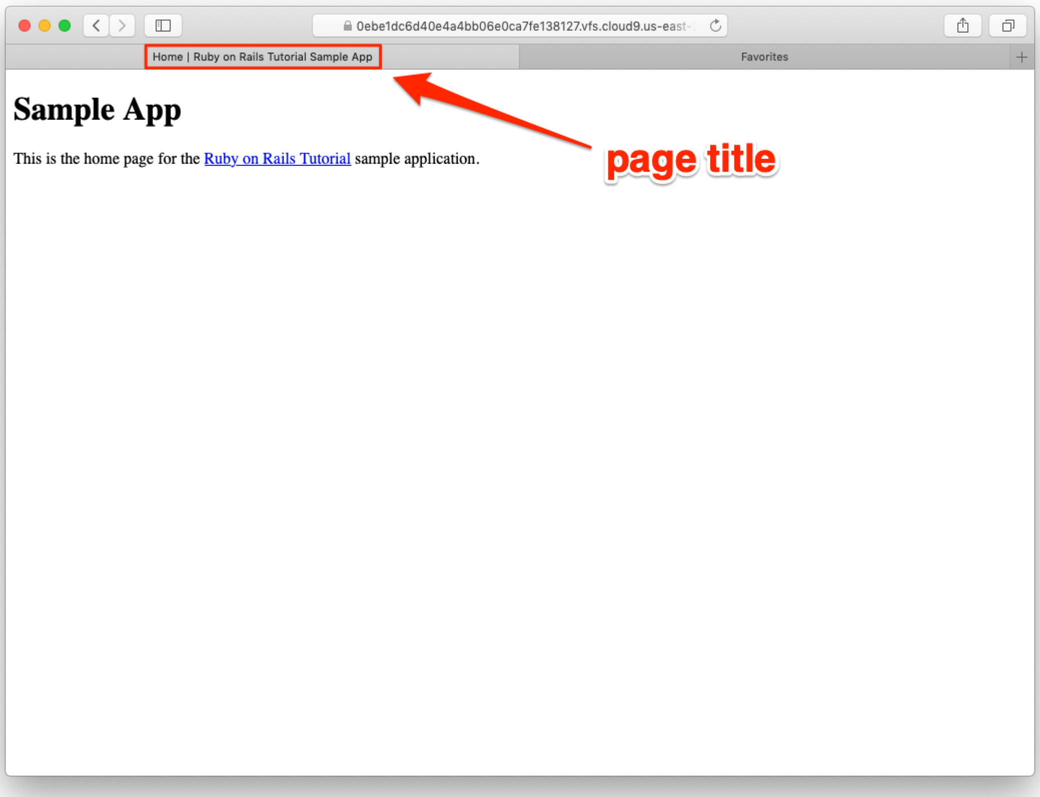
```
<!DOCTYPE html>
<html>
  <head>
    <title>Home | Ruby on Rails Tutorial Sample App</title>
  </head>
  <body>
    <h1>Sample App</h1>
    <p>
      This is the home page for the
      <a href="https://www.railstutorial.org/">Ruby on Rails Tutorial</a>
      sample application.
    </p>
  </body>
</html>
```

The corresponding web page appears in Figure 3.8. Note that the browser used in the screenshots (Safari) displays the page title only if you include an additional tab, which explains the second tab shown in Figure 3.8.

Following this model for the Help page (Listing 3.13) and the About page (Listing 3.23) yields the code in Listing 3.29 and Listing 3.30.

Listing 3.29: The view for the Help page with full HTML structure. RED
app/views/static_pages/help.html.erb

```
<!DOCTYPE html>
<html>
  <head>
    <title>Help | Ruby on Rails Tutorial Sample App</title>
  </head>
  <body>
    <h1>Help</h1>
    <p>
      Get help on the Ruby on Rails Tutorial at the
      <a href="https://www.railstutorial.org/help">Rails Tutorial help
      page</a>.
      To get help on this sample app, see the
      <a href="https://www.railstutorial.org/book"><em>Ruby on Rails
      Tutorial</em> book</a>.
    </p>
  </body>
</html>
```


0ebe1dc6d40e4a4bb06e0ca7fe138127.vfs.cloud9.us-east-
Home | Ruby on Rails Tutorial Sample App
Favorites
page title

Sample App

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

Figure 3.8: The Home page with a title.

Listing 3.30: The view for the About page with full HTML structure. GREEN
app/views/static_pages/about.html.erb

```
<!DOCTYPE html>
<html>
  <head>
    <title>About | Ruby on Rails Tutorial Sample App</title>
  </head>
  <body>
    <h1>About</h1>
    <p>
      The <a href="https://www.railstutorial.org/"><em>Ruby on Rails
      Tutorial</em></a>, part of the
      <a href="https://www.learnenough.com/">Learn Enough</a> family of
      tutorials, is a
      <a href="https://www.railstutorial.org/book">book</a> and
      <a href="https://screencasts.railstutorial.org/">screencast series</a>
      to teach web development with
      <a href="https://rubyonrails.org/">Ruby on Rails</a>.
      This is the sample app for the tutorial.
    </p>
  </body>
</html>
```

At this point, the test suite should be back to GREEN:

Listing 3.31: GREEN

```
$ rails test
3 tests, 6 assertions, 0 failures, 0 errors, 0 skips
```

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.

Beginning in this section, we'll start making modifications to the applications in the exercises that won't generally be reflected in future code listings. The reason is so that the text makes sense to readers who don't complete the exercises, but as a result your code will diverge from the main text if you *do* solve them. Learning to resolve small discrepancies like this is an excellent example of technical sophistication (Box 1.2).

1. You may have noticed some repetition in the Static Pages controller test (Listing 3.26). In particular, the base title, “Ruby on Rails Tutorial Sample App”, is the same for every title test. Using the special function **`setup`**, which is automatically run before every test, verify that the tests in Listing 3.32 are still GREEN. (Listing 3.32 uses an *instance variable*, seen briefly in Section 2.2.2 and covered further in Section 4.4.5, combined with *string interpolation*, which is covered further in Section 4.2.1.)

Listing 3.32: The Static Pages controller test with a base title. GREEN
`test/controllers/static_pages_controller_test.rb`

```
require 'test_helper'

class StaticPagesControllerTest < ActionDispatch::IntegrationTest

  def setup
    @base_title = "Ruby on Rails Tutorial Sample App"
  end

  test "should get home" do
    get static_pages_home_url
    assert_response :success
    assert_select "title", "Home | #{@base_title}"
  end

  test "should get help" do
    get static_pages_help_url
    assert_response :success
    assert_select "title", "Help | #{@base_title}"
  end

  test "should get about" do
    get static_pages_about_url
    assert_response :success
    assert_select "title", "About | #{@base_title}"
  end
end
```

3.4.3 Layouts and embedded Ruby (Refactor)

We've achieved a lot already in this section, generating three valid pages using Rails controllers and actions, but they are purely static HTML and hence don't show off the power of Rails. Moreover, they suffer from terrible duplication:

- The page titles are almost (but not quite) exactly the same.
- "Ruby on Rails Tutorial Sample App" is common to all three titles.
- The entire HTML skeleton structure is repeated on each page.

This repeated code is a violation of the important "Don't Repeat Yourself" (DRY) principle; in this section we'll "DRY out our code" by removing the repetition. At the end, we'll re-run the tests from Section 3.4.2 to verify that the titles are still correct.

Paradoxically, we'll take the first step toward eliminating duplication by first adding some more: we'll make the titles of the pages, which are currently quite similar, match *exactly*. This will make it much simpler to remove all the repetition at a stroke.

The technique involves using *embedded Ruby* in our views. Since the Home, Help, and About page titles have a variable component, we'll use a special Rails function called **`provide`** to set a different title on each page. We can see how this works by replacing the literal title "Home" in the **`home.html.erb`** view with the code in Listing 3.33.

Listing 3.33: The view for the Home page with an embedded Ruby title. GREEN
app/views/static_pages/home.html.erb

```
<% provide(:title, "Home") %>
<!DOCTYPE html>
<html>
  <head>
    <title><%= yield(:title) %> | Ruby on Rails Tutorial Sample App</title>
  </head>
  <body>
    <h1>Sample App</h1>
    <p>
```

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

Listing 3.33 is our first example of embedded Ruby, also called *ERb* (or ERB). (Now you know why HTML views have the file extension **`.html.erb`**.) ERb is the primary template system for including dynamic content in web pages.[12] The code

```
<% provide(:title, "Home") %>
```

indicates using **`<% ... %>`** that Rails should call the **`provide`** function and associate the string **`"Home"`** with the label **`:title`**.[13] Then, in the title, we use the closely related notation **`<%= ... %>`** to insert the title into the template using Ruby's **`yield`** function:[14]

```
<title><%= yield(:title) %> | Ruby on Rails Tutorial Sample App</title>
```

(The distinction between the two types of embedded Ruby is that **`<% ... %>`** *executes* the code inside, while **`<%= ... %>`** executes it *and inserts* the result into the template.) The resulting page is exactly the same as before, only now the variable part of the title is generated dynamically by ERb.

We can verify that all this works by running the tests from Section 3.4.2 and see that they are still GREEN:

[12]There is a second popular template system called Haml (note: not "HAML"), which I personally love, but it's not *quite* standard enough for use in an introductory tutorial.

[13]Experienced Rails developers might have expected the use of **`content_for`** at this point, but it doesn't work well with the asset pipeline. The **`provide`** function is its replacement.

[14]If you've studied Ruby before, you might suspect that Rails is *yielding* the contents to a block, and your suspicion would be correct. But you don't need to know this to develop applications with Rails.

Listing 3.34: GREEN

```
$ rails test
3 tests, 6 assertions, 0 failures, 0 errors, 0 skips
```

Then we can make the corresponding replacements for the Help and About pages (Listing 3.35 and Listing 3.36).

Listing 3.35: The view for the Help page with an embedded Ruby title. GREEN
app/views/static_pages/help.html.erb

```
<% provide(:title, "Help") %>
<!DOCTYPE html>
<html>
  <head>
    <title><%= yield(:title) %> | Ruby on Rails Tutorial Sample App</title>
  </head>
  <body>
    <h1>Help</h1>
    <p>
      Get help on the Ruby on Rails Tutorial at the
      <a href="https://www.railstutorial.org/help">Rails Tutorial help
      section</a>.
      To get help on this sample app, see the
      <a href="https://www.railstutorial.org/book"><em>Ruby on Rails
      Tutorial</em> book</a>.
    </p>
  </body>
</html>
```

Listing 3.36: The view for the About page with an embedded Ruby title. GREEN
app/views/static_pages/about.html.erb

```
<% provide(:title, "About") %>
<!DOCTYPE html>
<html>
  <head>
    <title><%= yield(:title) %> | Ruby on Rails Tutorial Sample App</title>
  </head>
  <body>
    <h1>About</h1>
    <p>
```

```
      The <a href="https://www.railstutorial.org/"><em>Ruby on Rails
      Tutorial</em></a>, part of the
      <a href="https://www.learnenough.com/">Learn Enough</a> family of
      tutorials, is a
      <a href="https://www.railstutorial.org/book">book</a> and
      <a href="https://screencasts.railstutorial.org/">screencast series</a>
      to teach web development with
      <a href="https://rubyonrails.org/">Ruby on Rails</a>.
      This is the sample app for the tutorial.
    </p>
  </body>
</html>
```

Now that we've replaced the variable part of the page titles with ERb, each of our pages looks something like this:

```
<% provide(:title, "Page Title") %>
<!DOCTYPE html>
<html>
  <head>
    <title><%= yield(:title) %> | Ruby on Rails Tutorial Sample App</title>
  </head>
  <body>
    Contents
  </body>
</html>
```

In other words, all the pages are identical in structure, including the contents of the title tag, with the sole exception of the material inside the **`body`** tag.

In order to factor out this common structure, Rails comes with a special *layout* file called **`application.html.erb`**, which we renamed in the beginning of this section (Section 3.4) and which we'll now restore:

```
$ mv layout_file app/views/layouts/application.html.erb
```

To get the layout to work, we have to replace the default title with the embedded Ruby from the examples above:

```
<title><%= yield(:title) %> | Ruby on Rails Tutorial Sample App</title>
```

The resulting layout appears in Listing 3.37.

Listing 3.37: The sample application site layout. GREEN
app/views/layouts/application.html.erb

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= yield(:title) %> | Ruby on Rails Tutorial Sample App</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' %>
  </head>

  <body>
    <%= yield %>
  </body>
</html>
```

Note here the special line

```
<%= yield %>
```

This code is responsible for inserting the contents of each page into the layout. It's not important to know exactly how this works; what matters is that using this layout ensures that, for example, visiting the page /static_pages/home converts the contents of `home.html.erb` to HTML and then inserts it in place of `<%= yield %>`.

It's also worth noting that the default Rails layout includes several additional lines:

```
<%= csrf_meta_tags %>
<%= csp_meta_tag %>
<%= stylesheet_link_tag ... %>
<%= javascript_pack_tag "application", ... %>
```

This code arranges to include the application stylesheet and JavaScript, which are part of the asset pipeline (Section 5.2.1), together with the Rails method **`csp_meta_tag`**, which implements Content Security Policy (CSP) to mitigate cross-site scripting (XSS) attacks, and **`csrf_meta_tags`**, which mitigates cross-site request forgery (CSRF) attacks. (One huge advantage of using a mature framework like Rails is that it worries about such things so that we don't have to.)

Even though the tests are passing, there one detail left to deal with: the views in Listing 3.33, Listing 3.35, and Listing 3.36 are still filled with all the HTML structure included in the layout. Since it's redundant (and indeed leads to invalid HTML markup) we should remove it and leave only the interior contents. The resulting cleaned-up views appear in Listing 3.38, Listing 3.39, and Listing 3.40.

Listing 3.38: The Home page with HTML structure removed. GREEN
app/views/static_pages/home.html.erb

```
<% provide(:title, "Home") %>
<h1>Sample App</h1>
<p>
  This is the home page for the
  <a href="https://www.railstutorial.org/">Ruby on Rails Tutorial</a>
  sample application.
</p>
```

Listing 3.39: The Help page with HTML structure removed. GREEN
app/views/static_pages/help.html.erb

```
<% provide(:title, "Help") %>
<h1>Help</h1>
<p>
  Get help on the Ruby on Rails Tutorial at the
  <a href="https://www.railstutorial.org/help">Rails Tutorial Help page</a>.
  To get help on this sample app, see the
  <a href="https://www.railstutorial.org/book"><em>Ruby on Rails Tutorial</em>
  book</a>.
</p>
```

Listing 3.40: The About page with HTML structure removed. GREEN
app/views/static_pages/about.html.erb

```
<% provide(:title, "About") %>
<h1>About</h1>
<p>
  The <a href="https://www.railstutorial.org/"><em>Ruby on Rails
  Tutorial</em></a>, part of the
  <a href="https://www.learnenough.com/">Learn Enough</a> family of
  tutorials, is a
  <a href="https://www.railstutorial.org/book">book</a> and
  <a href="https://screencasts.railstutorial.org/">screencast series</a>
  to teach web development with
  <a href="https://rubyonrails.org/">Ruby on Rails</a>.
  This is the sample app for the tutorial.
</p>
```

With these views defined, the Home, Help, and About pages are the same as before, but they have much less duplication.

Experience shows that even fairly simple refactoring is error-prone and can easily go awry. This is one reason why having a good test suite is so valuable. Rather than double-checking every page for correctness—a procedure that isn't too hard early on but rapidly becomes unwieldy as an application grows—we can simply verify that the test suite is still GREEN:

Listing 3.41: GREEN

```
$ rails test
3 tests, 6 assertions, 0 failures, 0 errors, 0 skips
```

This isn't a *proof* that our code is still correct, but it greatly increases the probability, thereby providing a safety net to protect us against future bugs.

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. Make a Contact page for the sample app.[15] Following the model in Listing 3.17, first write a test for the existence of a page at the URL /static_-pages/contact by testing for the title "Contact | Ruby on Rails Tutorial Sample App". Get your test to pass by following the same steps as when making the About page in Section 3.3.3, including filling the Contact page with the content from Listing 3.42.

Listing 3.42: Code for a proposed Contact page.
app/views/static_pages/contact.html.erb

```
<% provide(:title, "Contact") %>
<h1>Contact</h1>
<p>
  Contact the Ruby on Rails Tutorial about the sample app at the
  <a href="https://www.railstutorial.org/contact">contact page</a>.
</p>
```

3.4.4 Setting the root route

Now that we've customized our site's pages and gotten a good start on the test suite, let's set the application's root route before moving on. As in Section 1.2.4 and Section 2.2.2, this involves editing the **`routes.rb`** file to connect / to a page of our choice, which in this case will be the Home page. (At this point, I also recommend removing the **`hello`** action from the Application controller if you added it in Section 3.1.) As shown in Listing 3.43, this means changing the **`root`** route from

[15]This exercise is solved in Section 5.3.1.

```
root 'application#hello'
```

to

```
root 'static_pages#home'
```

This arranges for requests for / to be routed to the **home** action in the Static Pages controller. The resulting routes file is shown in Figure 3.9.

Listing 3.43: Setting the root route to the Home page.
config/routes.rb

```
Rails.application.routes.draw do
  root 'static_pages#home'
  get  'static_pages/home'
  get  'static_pages/help'
  get  'static_pages/about'
end
```

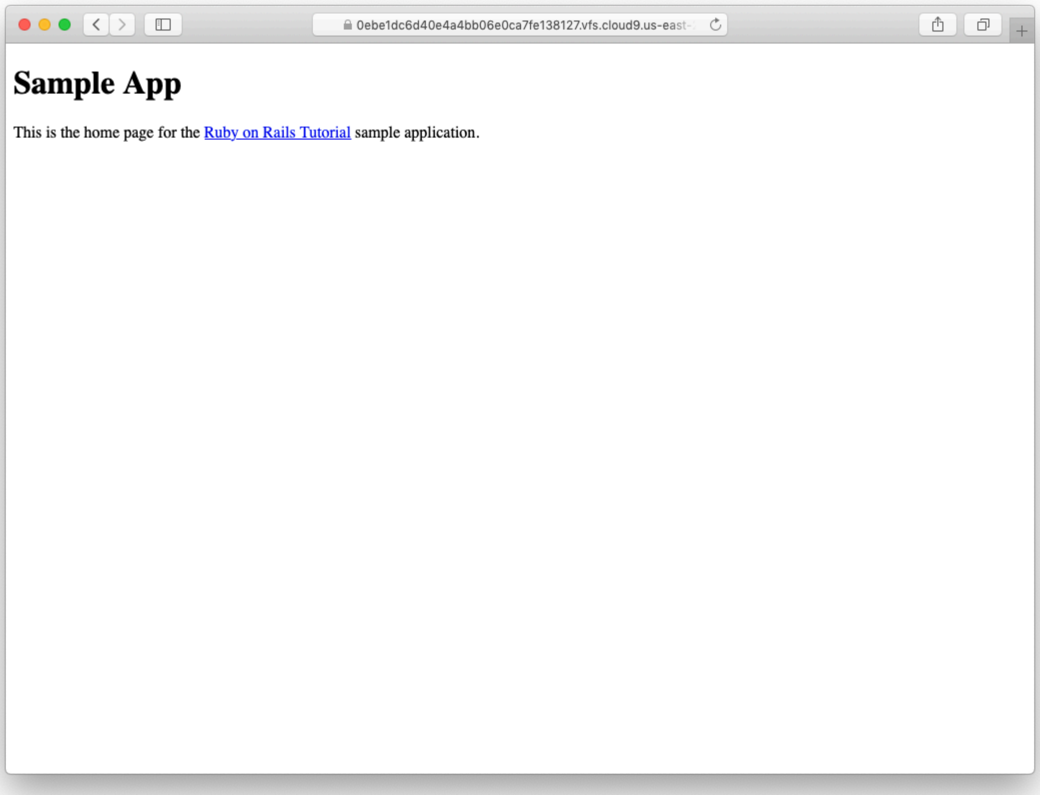
Exercises

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. Adding the root route in Listing 3.43 leads to the creation of a Rails helper called **root_url** (in analogy with helpers like **static_pages_home_url**). By filling in the code marked **FILL_IN** in Listing 3.44, write a test for the root route.

2. Due to the code in Listing 3.43, the test in the previous exercise is already GREEN. In such a case, it's harder to be confident that we're actually testing what we think we're testing, so modify the code in Listing 3.43 by commenting out the root route to get to RED (Listing 3.45). (We'll talk more about Ruby comments in Section 4.2.) Then uncomment it (thereby restoring the original Listing 3.43) and verify that you get back to GREEN.



Sample App

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

Figure 3.9: The Home page at the root route.

Listing 3.44: A test for the root route. GREEN
test/controllers/static_pages_controller_test.rb

```
require 'test_helper'

class StaticPagesControllerTest < ActionDispatch::IntegrationTest

  test "should get root" do
    get FILL_IN
    assert_response FILL_IN
  end

  test "should get home" do
    get static_pages_home_url
    assert_response :success
  end

  test "should get help" do
    get static_pages_help_url
    assert_response :success
  end

  test "should get about" do
    get static_pages_about_url
    assert_response :success
  end
end
```

Listing 3.45: Commenting out the root route to get a failing test. RED
config/routes.rb

```
Rails.application.routes.draw do
#   root 'static_pages#home'
  get  'static_pages/home'
  get  'static_pages/help'
  get  'static_pages/about'
end
```

3.5 Conclusion

Seen from the outside, this chapter hardly accomplished anything: we started with static pages, and ended with... *mostly* static pages. But appearances are