Page	URL	Named route
Home	/	`root_path`
About	/about	`about_path`
Help	/help	`help_path`
Contact	/contact	`contact_path`
Sign up	/signup	`signup_path`
Log in	/login	`login_path`

Table 5.1: Route and URL mapping for site links.

Listing 5.21: A test for the Contact page. 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", "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

  test "should get contact" do
    get static_pages_contact_url
    assert_response :success
    assert_select "title", "Contact | Ruby on Rails Tutorial Sample App"
  end
end
```

At this point, the tests in Listing 5.21 should be RED:

Listing 5.22: RED

```
$ rails test
```

The application code parallels the addition of the About page in Section 3.3: first we update the routes (Listing 5.23), then we add a **`contact`** action to the Static Pages controller (Listing 5.24), and finally we create a Contact view (Listing 5.25).

Listing 5.23: Adding a route for the Contact page. 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'
  get  'static_pages/contact'
end
```

Listing 5.24: Adding an action for the Contact page. RED
app/controllers/static_pages_controller.rb

```
class StaticPagesController < ApplicationController
  .
  .
  .
  def contact
  end
end
```

Listing 5.25: The view for the Contact page. GREEN
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>
```

Now make sure that the tests are GREEN:

Listing 5.26: GREEN

```
$ rails test
```

5.3.2 Rails routes

To add the named routes for the sample app's static pages, we'll edit the routes file, **`config/routes.rb`**, that Rails uses to define URL mappings. We'll begin by reviewing the route for the Home page (defined in Section 3.4.4), which is a special case, and then define a set of routes for the remaining static pages.

So far, we've seen three examples of how to define a root route, starting with the code

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

in the hello app (Listing 1.11), the code

```
root 'users#index'
```

in the toy app (Listing 2.7), and the code

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

in the sample app (Listing 3.43). In each case, the **`root`** method arranges for the root path / to be routed to a controller and action of our choice. Defining the root route in this way has a second important effect, which is to create named routes that allow us to refer to routes by a name rather than by the raw URL. In this case, these routes are **`root_path`** and **`root_url`**, with the only difference being that the latter includes the full URL:

```
root_path -> '/'
root_url  -> 'http://www.example.com/'
```

In the *Rails Tutorial*, we'll follow the common convention of using the **_path** form except when doing redirects, where we'll use the **_url** form. (This is because the HTTP standard technically requires a full URL after redirects, though in most browsers it will work either way.)

Because the default routes used in, e.g., Listing 5.21 are rather verbose, we'll also take this opportunity to define shorter named routes for the Help, About, and Contact pages. To do this, we need to make changes to the **get** rules from Listing 5.23, transforming lines like

```
get 'static_pages/help'
```

to

```
get  '/help', to: 'static_pages#help'
```

This new pattern routes a `GET` request for the URL /help to the **help** action in the Static Pages controller. As with the rule for the root route, this creates two named routes, **help_path** and **help_url**:

```
help_path -> '/help'
help_url  -> 'http://www.example.com/help'
```

Applying this rule change to the remaining static page routes from Listing 5.23 gives Listing 5.27.

Listing 5.27: Routes for static pages. RED

config/routes.rb

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

Note that Listing 5.27 also removes the route for `'static_pages/home'`, as we'll always use `root_path` or `root_url` instead.

Because the tests in Listing 5.21 used the old routes, they are now RED. To get them GREEN again, we need to update the routes as shown in Listing 5.28. Note that we've taken this opportunity to update to the (optional) convention of using the `*_path` form of each named route.

Listing 5.28: The static pages tests with the new named routes. GREEN
test/controllers/static_pages_controller_test.rb

```
require 'test_helper'

class StaticPagesControllerTest < ActionDispatch::IntegrationTest

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

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

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

  test "should get contact" do
    get contact_path
    assert_response :success
    assert_select "title", "Contact | Ruby on Rails Tutorial Sample App"
  end
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. It's possible to use a named route other than the default using the **as:** option. Drawing inspiration from this famous *Far Side* comic strip, change the route for the Help page to use **helf** (Listing 5.29).
2. Confirm that the tests are now RED. Get them to GREEN by updating the route in Listing 5.28.
3. Revert the changes from these exercises using Undo.

Listing 5.29: Changing 'help' to 'helf'.

```
Rails.application.routes.draw do
  root 'static_pages#home'
  get  '/help',    to: 'static_pages#help', as: 'helf'
  get  '/about',   to: 'static_pages#about'
  get  '/contact', to: 'static_pages#contact'
end
```

5.3.3 Using named routes

With the routes defined in Listing 5.27, we're now in a position to use the resulting named routes in the site layout. This simply involves filling in the second arguments of the **link_to** functions with the proper named routes. For example, we'll convert

```
<%= link_to "About", '#' %>
```

to

```
<%= link_to "About", about_path %>
```

and so on.

We'll start in the header partial, **_header.html.erb** (Listing 5.30), which has links to the Home and Help pages. While we're at it, we'll follow a common web convention and link the logo to the Home page as well.

Listing 5.30: Header partial with links.
app/views/layouts/_header.html.erb

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

We won't have a named route for the "Log in" link until Chapter 8, so we've left it as `'#'` for now.

The other place with links is the footer partial, **`_footer.html.erb`**, which has links for the About and Contact pages (Listing 5.31).

Listing 5.31: Footer partial with links.
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",   about_path %></li>
      <li><%= link_to "Contact", contact_path %></li>
      <li><a href="https://news.railstutorial.org/">News</a></li>
    </ul>
  </nav>
</footer>
```

With that, our layout has links to all the static pages created in Chapter 3, so that, for example, /about goes to the About page (Figure 5.10).

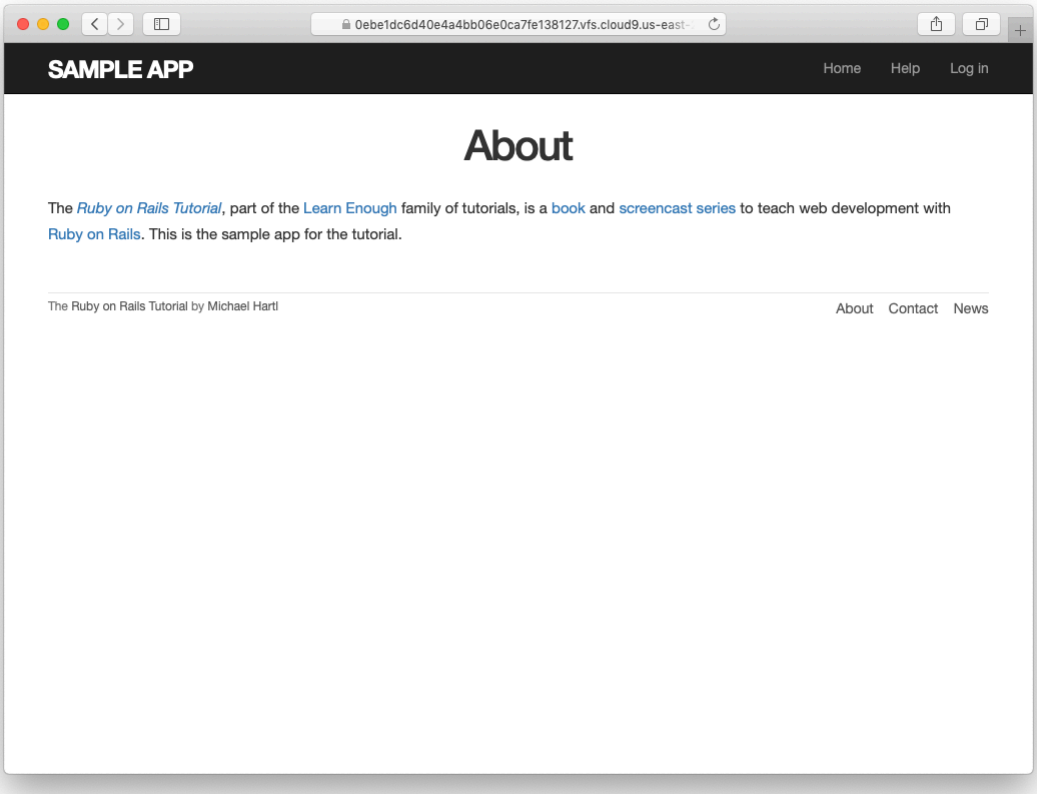

Figure 5.10: The About page at /about.

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. Update the layout links to use the `helf` route from Listing 5.29.
2. Revert the changes using Undo.

5.3.4 Layout link tests

Now that we've filled in several of the layout links, it's a good idea to test them to make sure they're working correctly. We could do this by hand with a browser, first visiting the root path and then checking the links by hand, but this quickly becomes cumbersome. Instead, we'll simulate the same series of steps using an *integration test*, which allows us to write an end-to-end test of our application's behavior. We can get started by generating a template test, which we'll call `site_layout`:

```
$ rails generate integration_test site_layout
      invoke  test_unit
      create    test/integration/site_layout_test.rb
```

Note that the Rails generator automatically appends `_test` to the name of the test file.

Our plan for testing the layout links involves checking the HTML structure of our site:

1. Get the root path (Home page).
2. Verify that the right page template is rendered.
3. Check for the correct links to the Home, Help, About, and Contact pages.

Listing 5.32 shows how we can use Rails integration tests to translate these steps into code, beginning with the **assert_template** method to verify that the Home page is rendered using the correct view.[18]

Listing 5.32: A test for the links on the layout. GREEN
test/integration/site_layout_test.rb

```
require 'test_helper'

class SiteLayoutTest < ActionDispatch::IntegrationTest

  test "layout links" do
    get root_path
    assert_template 'static_pages/home'
    assert_select "a[href=?]", root_path, count: 2
    assert_select "a[href=?]", help_path
    assert_select "a[href=?]", about_path
    assert_select "a[href=?]", contact_path
  end
end
```

Listing 5.32 uses some of the more advanced options of the **assert_select** method, seen before in Listing 3.26 and Listing 5.21. In this case, we use a syntax that allows us to test for the presence of a particular link–URL combination by specifying the tag name **a** and attribute **href**, as in

```
assert_select "a[href=?]", about_path
```

Here Rails automatically inserts the value of **about_path** in place of the question mark (escaping any special characters if necessary), thereby checking for an HTML tag of the form

[18] Some developers insist that a single test shouldn't contain multiple assertions. I find this practice to be unnecessarily complicated, while also incurring an extra overhead if there are common setup tasks needed before each test. In addition, a well-written test tells a coherent story, and breaking it up into individual pieces disrupts the narrative. I thus have a strong preference for including multiple assertions in a test, relying on Ruby (via minitest) to tell me the exact lines of any failed assertions.

Code	Matching HTML
`assert_select "div"`	`<div>foobar</div>`
`assert_select "div", "foobar"`	`<div>foobar</div>`
`assert_select "div.nav"`	`<div class="nav">foobar</div>`
`assert_select "div#profile"`	`<div id="profile">foobar</div>`
`assert_select "div[name=yo]"`	`<div name="yo">hey</div>`
`assert_select "a[href=?]", '/', count: 1`	`<a href="/">foo</a>`
`assert_select "a[href=?]", '/', text: "foo"`	`<a href="/">foo</a>`

Table 5.2: Some uses of `assert_select`.

```
<a href="/about">...</a>
```

Note that the assertion for the root path verifies that there are *two* such links (one each for the logo and navigation menu element):

```
assert_select "a[href=?]", root_path, count: 2
```

This ensures that both links to the Home page defined in Listing 5.30 are present.

Some more uses of `assert_select` appear in Table 5.2. While `assert_select` is flexible and powerful (having many more options than the ones shown here), experience shows that it's wise to take a lightweight approach by testing only HTML elements (such as site layout links) that are unlikely to change much over time.

To check that the new test in Listing 5.32 passes, we can run just the integration tests using the following Rake task:

Listing 5.33: GREEN

```
$ rails test:integration
```

If all went well, you should run the full test suite to verify that all the tests are GREEN:

Listing 5.34: GREEN

```
$ rails test
```

With the added integration test for layout links, we are now in a good position to catch regressions quickly using our test suite.

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. In the footer partial, change **`about_path`** to **`contact_path`** and verify that the tests catch the error.

2. It's convenient to use the **`full_title`** helper in the tests by including the Application helper into the test helper, as shown in Listing 5.35. We can then test for the right title using code like Listing 5.36. This is brittle, though, because now any typo in the base title (such as "Ruby on Rails Tutoial") won't be caught by the test suite. Fix this problem by writing a direct test of the **`full_title`** helper, which involves creating a file to test the application helper and then filling in the code indicated with **`FILL_IN`** in Listing 5.37. (Listing 5.37 uses **`assert_equal <expected>, <actual>`**, which verifies that the expected result matches the actual value when compared with the **`==`** operator.)

Listing 5.35: Including the Application helper in tests.
test/test_helper.rb

```
ENV['RAILS_ENV'] ||= 'test'
.
.
.
class ActiveSupport::TestCase
  fixtures :all
```

```
  include ApplicationHelper
  .
  .
  .
end
```

Listing 5.36: Using the `full_title` helper in a test. GREEN
test/integration/site_layout_test.rb

```
require 'test_helper'

class SiteLayoutTest < ActionDispatch::IntegrationTest

  test "layout links" do
    get root_path
    assert_template 'static_pages/home'
    assert_select "a[href=?]", root_path, count: 2
    assert_select "a[href=?]", help_path
    assert_select "a[href=?]", about_path
    assert_select "a[href=?]", contact_path
    get contact_path
    assert_select "title", full_title("Contact")
  end
end
```

Listing 5.37: A direct test of the `full_title` helper.
test/helpers/application_helper_test.rb

```
require 'test_helper'

class ApplicationHelperTest < ActionView::TestCase
  test "full title helper" do
    assert_equal full_title,         FILL_IN
    assert_equal full_title("Help"), FILL_IN
  end
end
```

5.4 User signup: A first step

As a capstone to our work on the layout and routing, in this section we'll make a route for the signup page, which will mean creating a second controller along