3. In Listing 6.35, we saw that email downcasing can be written more simply as `email.downcase!` (without any assignment). Make this change to the `downcase_email` method in Listing 11.3 and verify by running the test suite that it works.

11.2 Account activation emails

With the data modeling complete, we're now ready to add the code needed to send an account activation email. The method is to add a User *mailer* using the Action Mailer library, which we'll use in the Users controller `create` action to send an email with an activation link. Mailers are structured much like controller actions, with email templates defined as views. These templates will include links with the activation token and email address associated with the account to be activated.

11.2.1 Mailer templates

As with models and controllers, we can generate a mailer using `rails generate`, as shown in Listing 11.6.

Listing 11.6: Generating the User mailer.

```
$ rails generate mailer UserMailer account_activation password_reset
```

In addition to the necessary `account_activation` method, Listing 11.6 generates the `password_reset` method we'll need in Chapter 12.

The command in Listing 11.6 also generates two view templates for each mailer, one for plain-text email and one for HTML email. For the account activation mailer method, they appear as in Listing 11.7 and Listing 11.8. (We'll take care of the corresponding password reset templates in Chapter 12.)

Listing 11.7: The generated account activation text view.
app/views/user_mailer/account_activation.text.erb

```
UserMailer#account_activation

<%= @greeting %>, find me in app/views/user_mailer/account_activation.text.erb
```

Listing 11.8: The generated account activation HTML view.
app/views/user_mailer/account_activation.html.erb

```
<h1>UserMailer#account_activation</h1>

<p>
  <%= @greeting %>, find me in app/views/user_mailer/account_activation.html.erb
</p>
```

Let's take a look at the generated mailers to get a sense of how they work (Listing 11.9 and Listing 11.10). We see in Listing 11.9 that there is a default **`from`** address common to all mailers in the application, and each method in Listing 11.10 has a recipient's address as well. (Listing 11.9 also uses a mailer layout corresponding to the email format; although it won't ever matter in this tutorial, the resulting HTML and plain-text mailer layouts can be found in **`app/views/layouts`**.) The generated code also includes an instance variable (**`@greeting`**), which is available in the mailer views in much the same way that instance variables in controllers are available in ordinary views.

Listing 11.9: The generated application mailer.
app/mailers/application_mailer.rb

```
class ApplicationMailer < ActionMailer::Base
  default from: "from@example.com"
  layout 'mailer'
end
```

Listing 11.10: The generated User mailer.
app/mailers/user_mailer.rb

```
class UserMailer < ApplicationMailer

  # Subject can be set in your I18n file at config/locales/en.yml
  # with the following lookup:
  #
  #   en.user_mailer.account_activation.subject
  #
  def account_activation
    @greeting = "Hi"

    mail to: "to@example.org"
  end

  # Subject can be set in your I18n file at config/locales/en.yml
  # with the following lookup:
  #
  #   en.user_mailer.password_reset.subject
  #
  def password_reset
    @greeting = "Hi"

    mail to: "to@example.org"
  end
end
```

To make a working activation email, we'll first customize the generated template as shown in Listing 11.11. Next, we'll create an instance variable containing the user (for use in the view), and then mail the result to **`user.email`** (Listing 11.12). As seen in Listing 11.12, the **`mail`** method also takes a **`subject`** key, whose value is used as the email's subject line.

Listing 11.11: The application mailer with a new default **`from`** address.
app/mailers/application_mailer.rb

```
class ApplicationMailer < ActionMailer::Base
  default from: "noreply@example.com"
  layout 'mailer'
end
```

Listing 11.12: Mailing the account activation link. RED
app/mailers/user_mailer.rb

```
class UserMailer < ApplicationMailer

  def account_activation(user)
    @user = user
    mail to: user.email, subject: "Account activation"
  end

  def password_reset
    @greeting = "Hi"

    mail to: "to@example.org"
  end
end
```

As indicated in the Listing 11.12 caption, the tests are currently RED (due to our changing **`account_activation`** to take an argument); we'll get them to GREEN in Section 11.2.3.

As with ordinary views, we can use embedded Ruby to customize the template views, in this case greeting the user by name and including a link to a custom activation link. Our plan is to find the user by email address and then authenticate the activation token, so the link needs to include both the email and the token. Because we're modeling activations using an Account Activations resource, the token itself can appear as the argument of the named route defined in Listing 11.1:

```
edit_account_activation_url(@user.activation_token, ...)
```

Recalling that

```
edit_user_url(user)
```

produces a URL of the form

```
http://www.example.com/users/1/edit
```

the corresponding account activation link's base URL will look like this:

```
http://www.example.com/account_activations/q5lt38hQDc_959PVoo6b7A/edit
```

Here **`q5lt38hQDc_959PVoo6b7A`** is a URL-safe base64 string generated by the **`new_token`** method (Listing 9.2), and it plays the same role as the user id in /users/1/edit. In particular, in the Activations controller **`edit`** action, the token will be available in the **`params`** hash as **`params[:id]`**.

In order to include the email as well, we need to use a *query parameter*, which in a URL appears as a key-value pair located after a question mark:[6]

```
account_activations/q5lt38hQDc_959PVoo6b7A/edit?email=foo%40example.com
```

Notice that the '@' in the email address appears as **`%40`**, i.e., it's "escaped out" to guarantee a valid URL. The way to set a query parameter in Rails is to include a hash in the named route:

```
edit_account_activation_url(@user.activation_token, email: @user.email)
```

When using named routes in this way to define query parameters, Rails automatically escapes out any special characters. The resulting email address will also be unescaped automatically in the controller, and will be available via **`params[:email]`**.

With the **`@user`** instance variable as defined in Listing 11.12, we can create the necessary links using the named edit route and embedded Ruby, as shown in Listing 11.13 and Listing 11.14. Note that the HTML template in Listing 11.14 uses the **`link_to`** method to construct a valid link.

[6] URLs can contain multiple query parameters, consisting of multiple key-value pairs separated by the ampersand character **`&`**, as in **`/edit?name=Foo%20Bar&email=foo%40example.com`**.

Listing 11.13: The account activation text view.
app/views/user_mailer/account_activation.text.erb

```
Hi <%= @user.name %>,

Welcome to the Sample App! Click on the link below to activate your account:

<%= edit_account_activation_url(@user.activation_token, email: @user.email) %>
```

Listing 11.14: The account activation HTML view.
app/views/user_mailer/account_activation.html.erb

```
<h1>Sample App</h1>

<p>Hi <%= @user.name %>,</p>

<p>
Welcome to the Sample App! Click on the link below to activate your account:
</p>

<%= link_to "Activate", edit_account_activation_url(@user.activation_token,
                                                    email: @user.email) %>
```

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. At the console, verify that the **`escape`** method in the **`CGI`** module escapes out the email address as shown in Listing 11.15. What is the escaped value of the string **`"Don't panic!"`**?

Listing 11.15: Escaping an email with **`CGI.escape`**.

```
>> CGI.escape('foo@example.com')
=> "foo%40example.com"
```

11.2.2 Email previews

To see the results of the templates defined in Listing 11.13 and Listing 11.14, we can use *email previews*, which are special URLs exposed by Rails to let us see what our email messages look like. First, we need to add some configuration to our application's development environment, as shown in Listing 11.16.

Listing 11.16: Email settings in development.
config/environments/development.rb

```
Rails.application.configure do
  .
  .
  .
  config.action_mailer.raise_delivery_errors = false

  host = 'example.com' # Don't use this literally; use your local dev host instead
  # Use this on the cloud IDE.
  config.action_mailer.default_url_options = { host: host, protocol: 'https' }
  # Use this if developing on localhost.
  # config.action_mailer.default_url_options = { host: host, protocol: 'http' }
  .
  .
  .
end
```

Listing 11.16 uses a host name of `'example.com'`, but as indicated in the comment you should use the actual host of your development environment. For example, on the cloud IDE you should use

```
host = '<hex string>.vfs.cloud9.us-east-2.amazonaws.com'     # Cloud IDE
config.action_mailer.default_url_options = { host: host, protocol: 'https' }
```

where the exact URL is based on what you see in your browser (Figure 11.2).

On a local system, you should use this instead:

```
host = 'localhost:3000'                      # Local server
config.action_mailer.default_url_options = { host: host, protocol: 'http' }
```

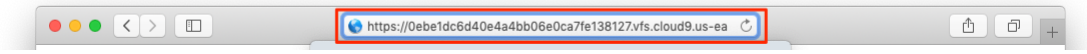

Figure 11.2: The host URL for the cloud IDE.

Note especially in this second example that **`https`** has changed to plain **`http`**.

After restarting the development server to activate the configuration in Listing 11.16, we next need to update the User mailer *preview file*, which was automatically generated in Section 11.2, as shown in Listing 11.17.

Listing 11.17: The generated User mailer previews.
test/mailers/previews/user_mailer_preview.rb

```
# Preview all emails at http://localhost:3000/rails/mailers/user_mailer
class UserMailerPreview < ActionMailer::Preview

  # Preview this email at
  # http://localhost:3000/rails/mailers/user_mailer/account_activation
  def account_activation
    UserMailer.account_activation
  end

  # Preview this email at
  # http://localhost:3000/rails/mailers/user_mailer/password_reset
  def password_reset
    UserMailer.password_reset
  end

end
```

Because the **`account_activation`** method defined in Listing 11.12 requires a valid user object as an argument, the code in Listing 11.17 won't work as written. To fix it, we define a **`user`** variable equal to the first user in the development database, and then pass it as an argument to **`UserMailer.account_activation`** (Listing 11.18). Note that Listing 11.18 also assigns a value to **`user.activation_token`**, which is necessary because the account activation templates in Listing 11.13 and Listing 11.14 need an account activation token. (Because **`activation_token`** is a virtual attribute (Section 11.1), the user from the database doesn't have one.)

Listing 11.18: A working preview method for account activation.
test/mailers/previews/user_mailer_preview.rb

```
# Preview all emails at http://localhost:3000/rails/mailers/user_mailer
class UserMailerPreview < ActionMailer::Preview

  # Preview this email at
  # http://localhost:3000/rails/mailers/user_mailer/account_activation
  def account_activation
    user = User.first
    user.activation_token = User.new_token
    UserMailer.account_activation(user)
  end

  # Preview this email at
  # http://localhost:3000/rails/mailers/user_mailer/password_reset
  def password_reset
    UserMailer.password_reset
  end
end
```

With the preview code as in Listing 11.18, we can visit the suggested URLs to preview the account activation emails. (If you are using the cloud IDE, you should replace **`localhost:3000`** with the corresponding base URL.) The resulting HTML and text emails appear as in Figure 11.3 and Figure 11.4.

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. Preview the email templates in your browser. What do the Date fields read for your previews?

11.2.3 Email tests

As a final step, we'll write a couple of tests to double-check the results shown in the email previews. This isn't as hard as it sounds, because Rails has generated useful example tests for us (Listing 11.19).

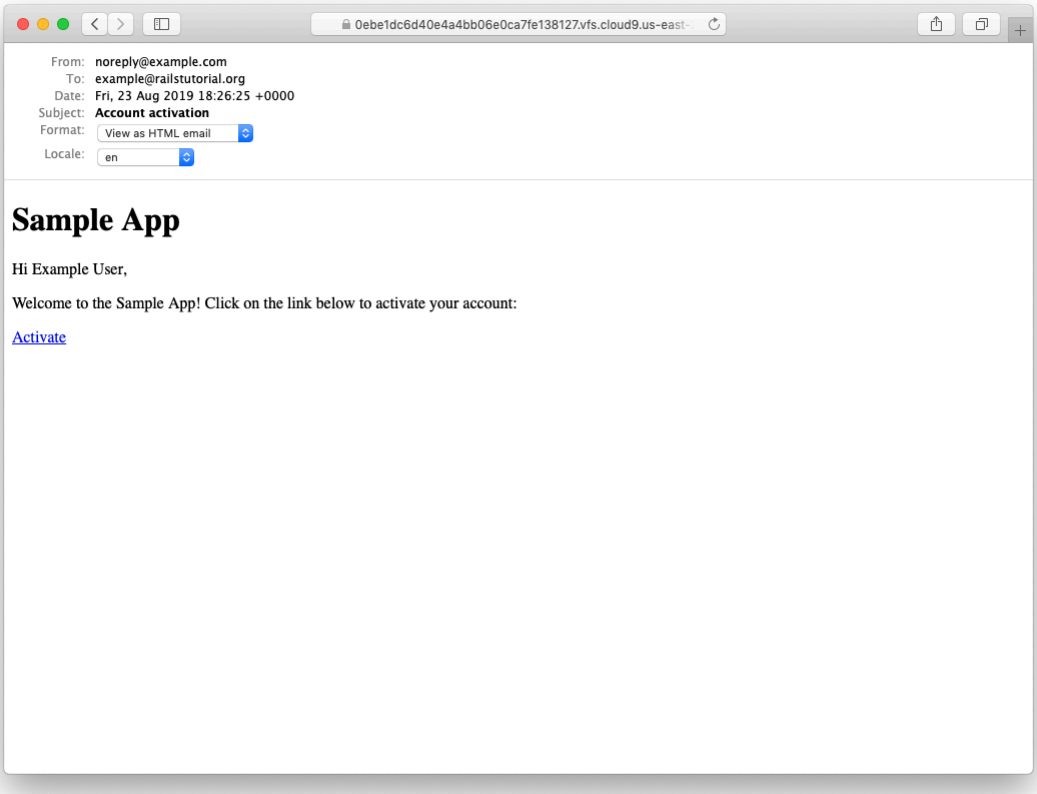

Figure 11.3: A preview of the HTML version of the account activation email.

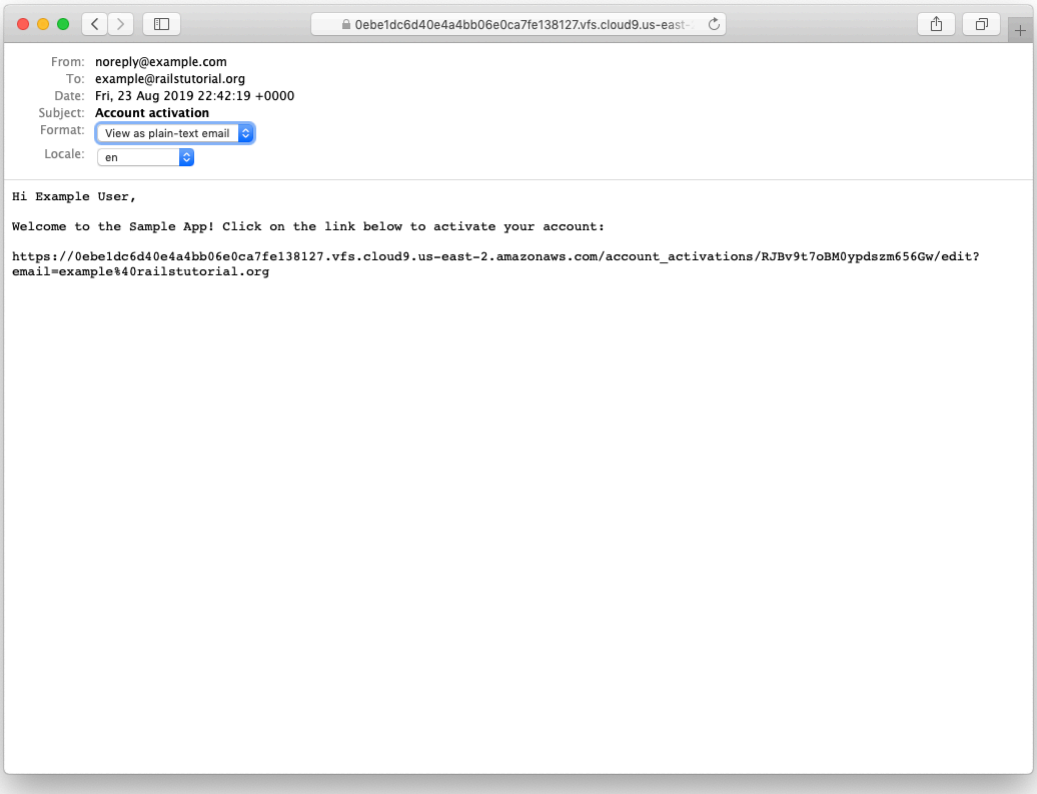

Figure 11.4: A preview of the text version of the account activation email.

Listing 11.19: The User mailer test generated by Rails. RED
test/mailers/user_mailer_test.rb

```
require 'test_helper'

class UserMailerTest < ActionMailer::TestCase

  test "account_activation" do
    mail = UserMailer.account_activation
    assert_equal "Account activation", mail.subject
    assert_equal ["to@example.org"], mail.to
    assert_equal ["from@example.com"], mail.from
    assert_match "Hi", mail.body.encoded
  end

  test "password_reset" do
    mail = UserMailer.password_reset
    assert_equal "Password reset", mail.subject
    assert_equal ["to@example.org"], mail.to
    assert_equal ["from@example.com"], mail.from
    assert_match "Hi", mail.body.encoded
  end
end
```

As mentioned in Section 11.2.1, the tests in Listing 11.19 are currently RED.

The tests in Listing 11.19 use the powerful **assert_match** method, which can be used either with a string or a regular expression:

```
assert_match 'foo', 'foobar'      # true
assert_match 'baz', 'foobar'      # false
assert_match /\w+/, 'foobar'      # true
assert_match /\w+/, '$#!*+@'      # false
```

The test in Listing 11.20 uses **assert_match** to check that the name, activation token, and escaped email appear in the email's body. For the last of these, note the use of

```
CGI.escape(user.email)
```

to escape the test user's email, which we met briefly in Section 11.2.1.[7]

[7] When I originally wrote this chapter, I couldn't recall offhand how to escape URLs in Rails, and figuring it

Listing 11.20: A test of the current email implementation. RED
test/mailers/user_mailer_test.rb

```
require 'test_helper'

class UserMailerTest < ActionMailer::TestCase

  test "account_activation" do
    user = users(:michael)
    user.activation_token = User.new_token
    mail = UserMailer.account_activation(user)
    assert_equal "Account activation", mail.subject
    assert_equal [user.email], mail.to
    assert_equal ["noreply@example.com"], mail.from
    assert_match user.name,               mail.body.encoded
    assert_match user.activation_token,   mail.body.encoded
    assert_match CGI.escape(user.email),  mail.body.encoded
  end
end
```

Note that Listing 11.20 takes care to add an activation token to the fixture user, which would otherwise be blank. (Listing 11.20 also removes the generated password reset test, which we'll add back (in modified form) in Section 12.2.2.)

To get the test in Listing 11.20 to pass, we have to configure our test file with the proper domain host, as shown in Listing 11.21.

Listing 11.21: Setting the test domain host. GREEN
config/environments/test.rb

```
Rails.application.configure do
  .
  .
  .
  config.action_mailer.delivery_method = :test
  config.action_mailer.default_url_options = { host: 'example.com' }
  .
  .
  .
end
```

out was pure technical sophistication (Box 1.2). What I did was Google "ruby rails escape url", which led me to find two main possibilities, `URI.encode(str)` and `CGI.escape(str)`. Trying them both revealed that the latter works. (It turns out there's a third possibility: the `ERB::Util` library supplies a url_encode method that has the same effect.)

With the code as above, the mailer test should be GREEN:

Listing 11.22: GREEN

```
$ rails test:mailers
```

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. Verify that the full test suite is still GREEN.
2. Confirm that the test goes RED if you remove the call to `CGI.escape` in Listing 11.20.

11.2.4 Updating the Users `create` action

To use the mailer in our application, we just need to add a couple of lines to the `create` action used to sign users up, as shown in Listing 11.23. Note that Listing 11.23 has changed the redirect behavior upon signing up. Before, we redirected to the user's profile page (Section 7.4), but that doesn't make sense now that we're requiring account activation. Instead, we now redirect to the root URL.

Listing 11.23: Adding account activation to user signup. RED
app/controllers/users_controller.rb

```
class UsersController < ApplicationController
  .
  .
  .
  def create
    @user = User.new(user_params)
    if @user.save
      UserMailer.account_activation(@user).deliver_now
```

```
      flash[:info] = "Please check your email to activate your account."
      redirect_to root_url
    else
      render 'new'
    end
  end
  .
  .
  .
end
```

Because Listing 11.23 redirects to the root URL instead of to the profile page and doesn't log the user in as before, the test suite is currently RED, even though the application is working as designed. We'll fix this by temporarily commenting out the failing lines, as shown in Listing 11.24. We'll uncomment these lines and write passing tests for account activation in Section 11.3.3.

Listing 11.24: Temporarily commenting out failing tests. GREEN
test/integration/users_signup_test.rb

```
require 'test_helper'

class UsersSignupTest < ActionDispatch::IntegrationTest

  test "invalid signup information" do
    get signup_path
    assert_no_difference 'User.count' do
      post users_path, params: { user: { name:  "",
                                         email: "user@invalid",
                                         password:              "foo",
                                         password_confirmation: "bar" } }
    end
    assert_template 'users/new'
    assert_select 'div#error_explanation'
    assert_select 'div.field_with_errors'
  end

  test "valid signup information" do
    get signup_path
    assert_difference 'User.count', 1 do
      post users_path, params: { user: { name:  "Example User",
                                         email: "user@example.com",
                                         password:              "password",
                                         password_confirmation: "password" } }
    end
    follow_redirect!
```

```
    # assert_template 'users/show'
    # assert is_logged_in?
  end
end
```

If you now try signing up as a new user, you should be redirected as shown in Figure 11.5, and an email like the one shown in Listing 11.25 should be generated. Note that you will *not* receive an actual email in a development environment, but it will show up in your server logs. (You may have to scroll up a bit to see it.) Section 11.4 discusses how to send email for real in a production environment.

Listing 11.25: A sample account activation email from the server log.

```
UserMailer#account_activation: processed outbound mail in 5.1ms
Delivered mail 5d606e97b7a44_28872b106582df988776a@ip-172-31-25-202.mail (3.2ms)
Date: Fri, 23 Aug 2019 22:54:15 +0000
From: noreply@example.com
To: michael@michaelhartl.com
Message-ID: <5d606e97b7a44_28872b106582df988776a@ip-172-31-25-202.mail>
Subject: Account activation
Mime-Version: 1.0
Content-Type: multipart/alternative;
 boundary="----=_mimepart_5d606e97b6f16_28872b106582df98876dd";
 charset=UTF-8
Content-Transfer-Encoding: 7bit


------=_mimepart_5d606e97b6f16_28872b106582df98876dd
Content-Type: text/plain;
 charset=UTF-8
Content-Transfer-Encoding: 7bit

Hi Michael Hartl,

Welcome to the Sample App! Click on the link below to activate your account:

https://0ebe1dc6d40e4a4bb06e0ca7fe138127.vfs.cloud9.us-east-2.
amazonaws.com/account_activations/zdqs6sF7BMiDfXBaC7-6vA/
edit?email=michael%40michaelhartl.com

------=_mimepart_5d606e97b6f16_28872b106582df98876dd
Content-Type: text/html;
 charset=UTF-8
Content-Transfer-Encoding: 7bit
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style>
      /* Email styles need to be inline */
    </style>
  </head>

  <body>
    <h1>Sample App</h1>

<p>Hi Michael Hartl,</p>

<p>
Welcome to the Sample App! Click on the link below to activate your account:
</p>

<a href="https://0ebe1dc6d40e4a4bb06e0ca7fe138127.vfs.cloud9.us-east-2.
amazonaws.com/account_activations/zdqs6sF7BMiDfXBaC7-6vA/
edit?email=michael%40michaelhartl.com">Activate</a>
  </body>
</html>

------==_mimepart_5d606e97b6f16_28872b106582df98876dd--
```

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. Sign up as a new user and verify that you're properly redirected. What is the content of the generated email in the server log? What is the value of the activation token?

2. Verify at the console that the new user has been created but that it is not yet activated.

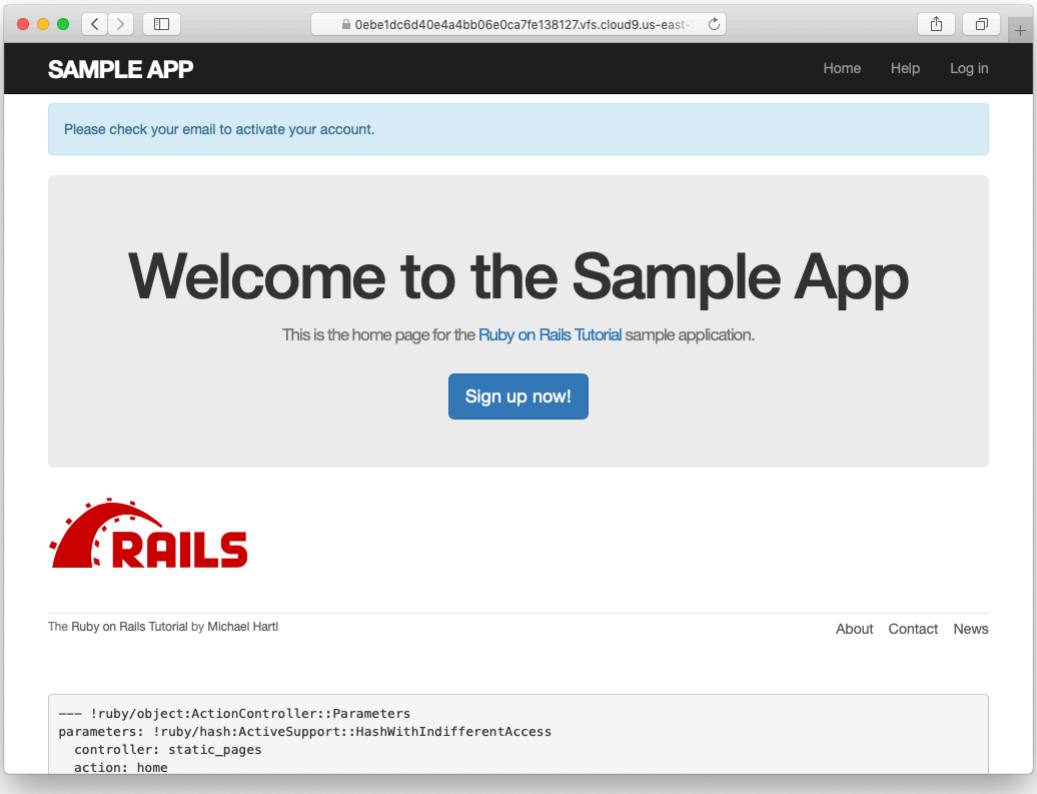

Figure 11.5: The Home page with an activation message after signup.