COVERS RAILS 4.2

THE RUBY ON RAILS TUTORIAL

SOLUTIONS MANUAL FOR EXERCISES

THIRD EDITION

BOOK AND SCREENCASTS BY MICHAEL HARTL

Ruby on Rails Tutorial

Solutions Manual for Exercises

Michael Hartl

Contents

About the author

Michael Hartl is the author of the *Ruby on Rails Tutorial*, one of the leading introductions to web development, and is a cofounder of the Softcover self-publishing platform. His prior experience includes writing and developing *RailsSpace*, an extremely obsolete Rails tutorial book, and developing Insoshi, a once-popular and now-obsolete social networking platform in Ruby on Rails. In 2011, Michael received a Ruby Hero Award for his contributions to the Ruby community. He is a graduate of Harvard College, has a Ph.D. in Physics from Caltech, and is an alumnus of the Y Combinator entrepreneur program.

Copyright and license

Chapter 1

Solutions to Chapter 1 exercises

This is the solutions manual for the Ruby on Rails Tutorial, with solutions to all the exercises in the *Ruby on Rails Tutorial* book. The original questions appear in the *Exercises* sections of each chapter; they are not reproduced here both due to technical restrictions (cross-references break and are difficult to update, for example) and because friction when using solutions is a feature, not a bug. In particular, I strongly encourage all readers to grapple with the exercises on their own before reading the solutions here. Finally, while every effort has been made to make these solutions clear and complete, errors (both typographical and otherwise) may have slipped through, and reports of such errors are gratefully received. Please send them to admin@railstutorial.org.

1.1 Exercise 1

The basic solution involves editing the **`hello`** action in the Application controller, as shown in Listing 1.1.[1]

[1]The lines above the **`hello`** action in Listing 1.1 are auto-generated by **`rails new`** and might differ depending on the version of Rails you're using.

Listing 1.1: Changing “hello, world!” to “hola, mundo!”.
app/controllers/application_controller.rb

```
class ApplicationController < ActionController::Base
  # Prevent CSRF attacks by raising an exception.
  # For APIs, you may want to use :null_session instead.
  protect_from_forgery with: :exception

  def hello
    render text: "hola, mundo!"
  end
end
```

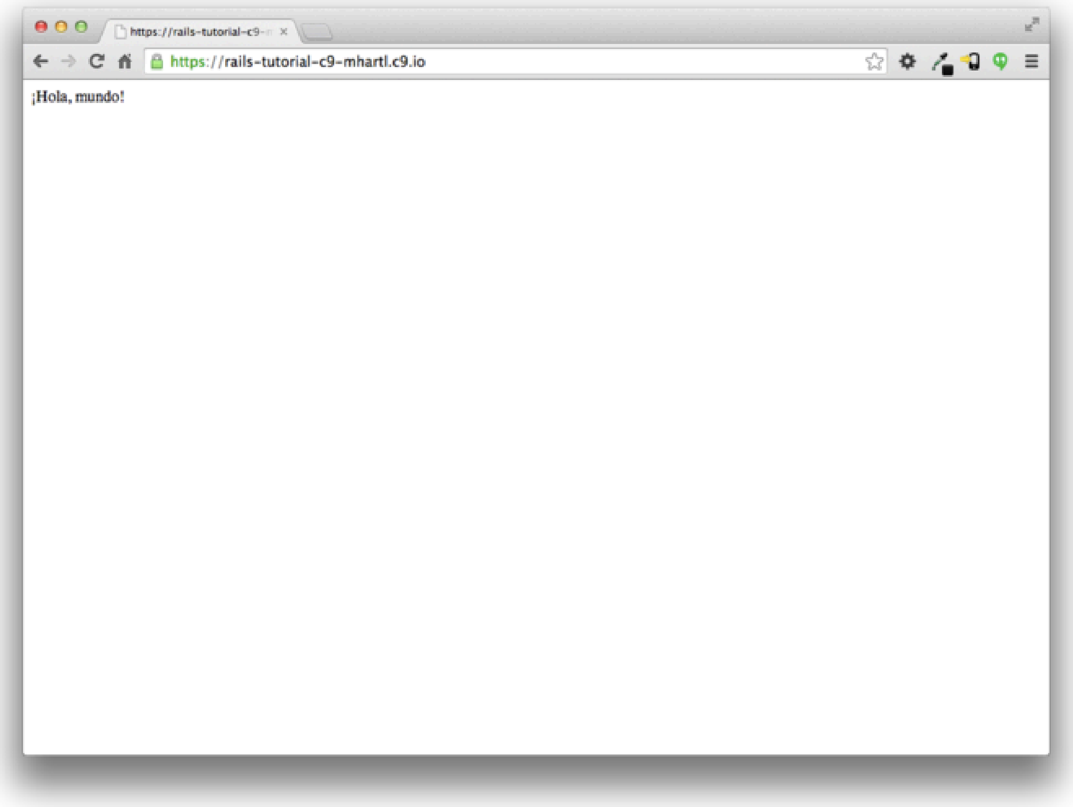
To solve the extra credit portion, you need to produce an inverted exclamation point “¡”. This can be done any number of ways, including copy-and-pasting from the *Ruby on Rails Tutorial* itself or by using a character insertion palette (whose availability may vary by system). If you’re using a Macintosh, you can use my favorite method by pressing Option-1, which makes it easy to go *¡¡¡¡¡¡totalmente loco!!!!!!* No matter how you do it, the result should appear as in Listing 1.2. (The text in Listing 1.2 capitalizes the first letter in “Hola”, because to my eye “¡hola, mundo!” looks strange.)[2]

Listing 1.2: Changing “hola, mundo!” to “¡Hola, mundo!”.
app/controllers/application_controller.rb

```
class ApplicationController < ActionController::Base
  # Prevent CSRF attacks by raising an exception.
  # For APIs, you may want to use :null_session instead.
  protect_from_forgery with: :exception

  def hello
    render text: "¡Hola, mundo!"
  end
end
```

The result appears in Figure 1.1.

[2]Arguably, “hello, world!” should look strange, too, but I’ve seen it in so many programming examples that by now it looks normal.


https://rails-tutorial-c9-m
https://rails-tutorial-c9-mhartl.c9.io

¡Hola, mundo!

Figure 1.1: Changing the root route to return "¡Hola, mundo!".

1.2 Exercise 2

Two changes are required to replace "hello, world!" (or "¡Hola, mundo!") with "goodbye, world!". The first is to add the **`goodbye`** action to the Application controller, as shown in Listing 1.3

Listing 1.3: Adding a **`goodbye`** action.
app/controllers/application_controller.rb

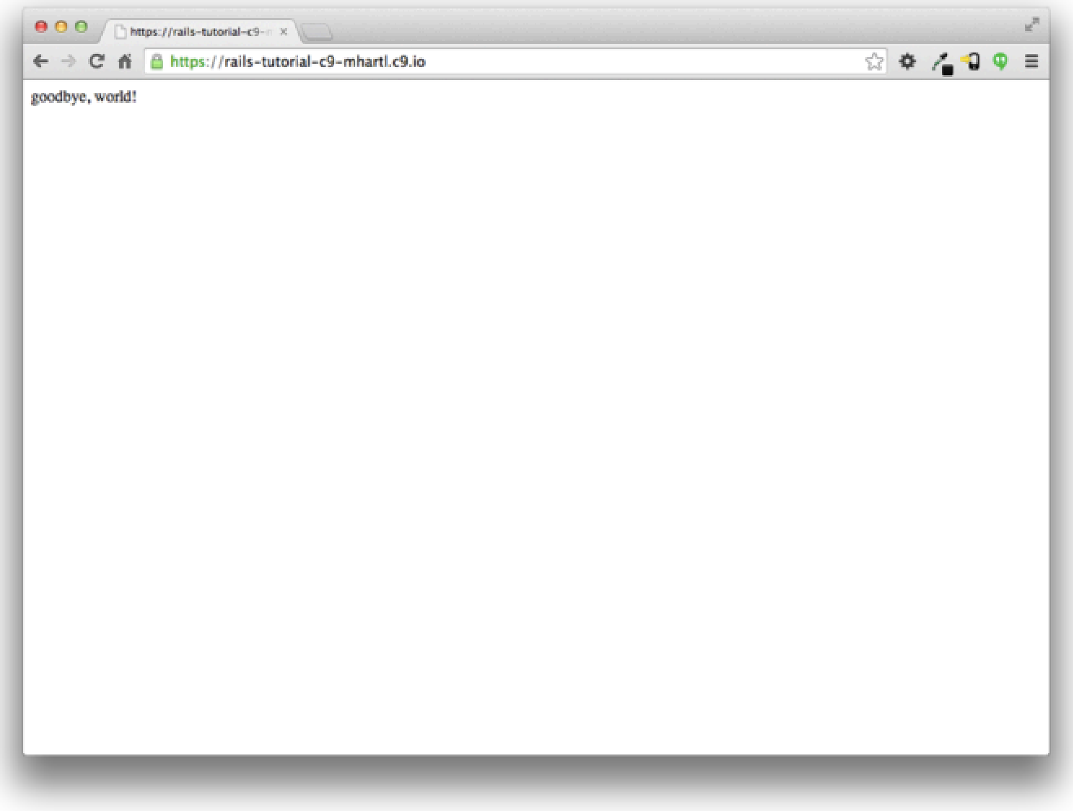
```ruby
class ApplicationController < ActionController::Base
  # Prevent CSRF attacks by raising an exception.
  # For APIs, you may want to use :null_session instead.
  protect_from_forgery with: :exception

  def hello
    render text: "¡Hola, mundo!"
  end

  def goodbye
    render text: "goodbye, world!"
  end
end
```

The second change is to edit **`routes.rb`** to update the root route so that it points to the **`goodbye`** action, as shown in Listing 1.4.

Listing 1.4: Setting the root route to say "goodbye, world!".
config/routes.rb

```ruby
Rails.application.routes.draw do
  .
  .
  .
  # You can have the root of your site routed with "root"
  root 'application#goodbye'
  .
  .
  .
end
```

The result appears in Figure 1.2.



goodbye, world!

Figure 1.2: Changing the root route to return "goodbye, world!".

Chapter 2

Solutions to Chapter 2 exercises

2.1 Exercise 1

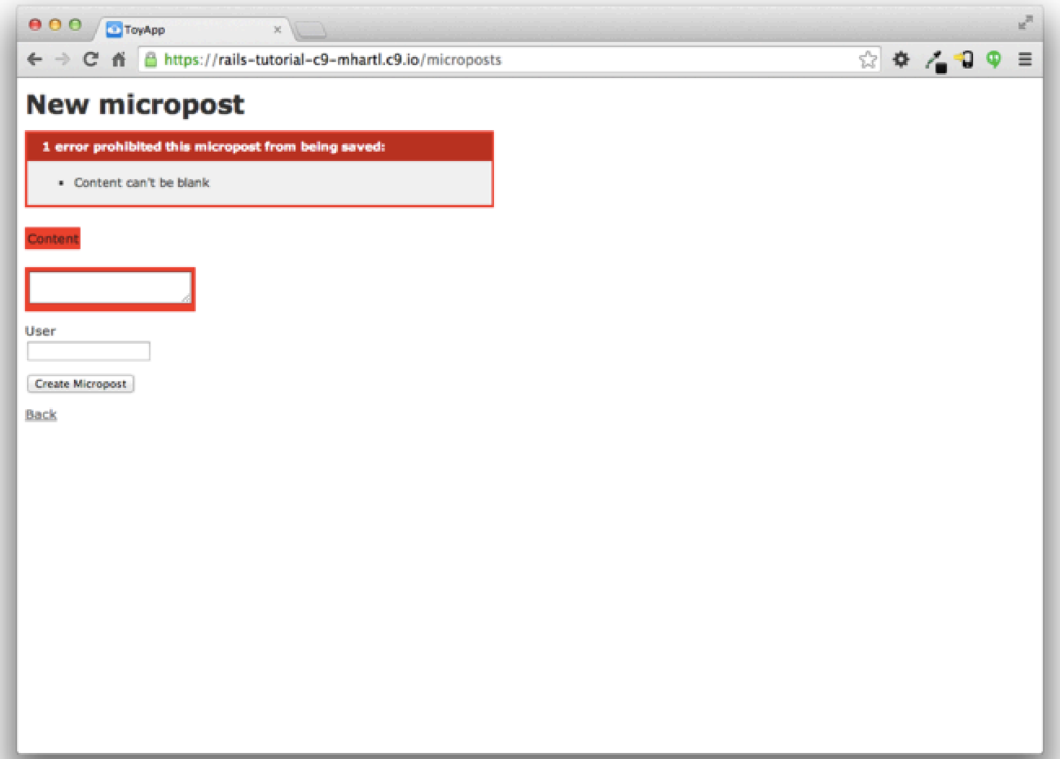
The solution was given in the exercise and appears as in Listing 2.1. If you submit the new micropost form with an empty content field, you should get the results shown in Figure 2.1.

Listing 2.1: Code to validate the presence of micropost content.
app/models/micropost.rb

```
class Micropost < ActiveRecord::Base
  belongs_to :user
  validates :content, length: { maximum: 140 },
                      presence: true
end
```

2.2 Exercise 2

Fill in the two occurrences of `FILL_IN` with the symbols `:name` and `:email`, respectively, as shown in Listing 2.2. The result should be as shown in Figure 2.2.



New micropost

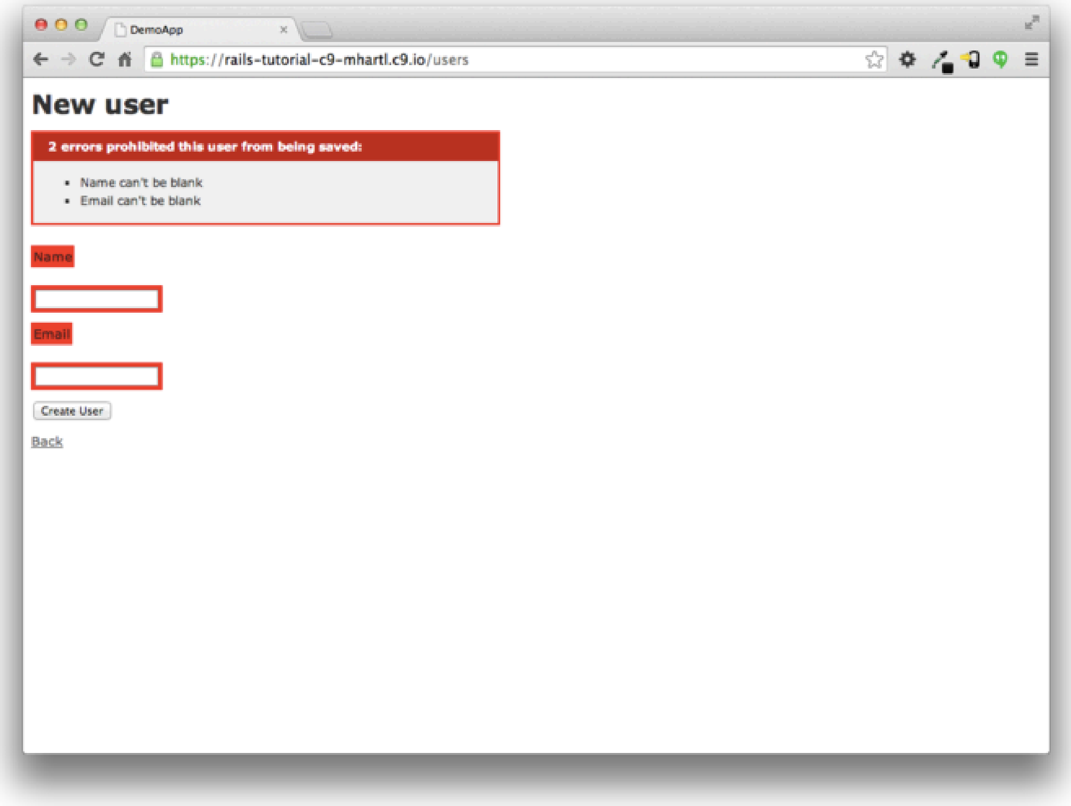
1 error prohibited this micropost from being saved:

- Content can't be blank

Content

User

Figure 2.1: The effect of a micropost presence validation.



New user

2 errors prohibited this user from being saved:

- Name can't be blank
- Email can't be blank

Figure 2.2: The effect of presence validations on the User model.

Listing 2.2: Adding presence validations to the User model.
app/models/user.rb

```ruby
class User < ActiveRecord::Base
  has_many :microposts
  validates :name,  presence: true
  validates :email, presence: true
end
```

Chapter 3

Solutions to Chapter 3 exercises

3.1 Exercise 1

The answer appears as part of the exercise, as shown in Listing 3.1.

Listing 3.1: The Static Pages controller test with a base title. **GREEN**
test/controllers/static_pages_controller_test.rb

```
require 'test_helper'

class StaticPagesControllerTest < ActionController::TestCase

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

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

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

You can verify that the tests defined by Listing 3.1 are **GREEN** as follows:

Listing 3.2: GREEN

```
$ bundle exec rake test
```

To eliminate duplication, Listing 3.1 uses the `setup` function, an *instance variable*, and *variable interpolation*. Inside a test file, the function called `setup` has a special meaning: it is automatically run before every test. (This means it's important to avoid putting time-consuming code inside `setup` if possible, as this can significantly slow down a test suite.) In Listing 3.1, the `setup` function defines a variable called `@base_title` (read "at base title"), which is identified as an instance variable by the @ sign at the beginning of the name. Instance variables have many uses in Ruby, but in this context the most important characteristic is that instance variables defined inside `setup` are available inside each test. In particular, if we define the base title using

```
@base_title = "Ruby on Rails Tutorial Sample App"
```

then the interpolated value

```
"#{@base_title}"
```

is equal to the title string

```
"Ruby on Rails Tutorial Sample App"
```

This means that, for example, the code

```
"Home | #{@base_title}"
```

is equal to the full title

```
"Home | Ruby on Rails Tutorial Sample App"
```

as required. (Notably, using a variable called `base_title`—*without* the @ sign—wouldn't work in the same context.)

3.2 Exercise 2

To add an About page using test-driven development, we begin with the test supplied as part of the exercise, as shown in Listing 3.3. (In order to keep the tests independent, Listing 3.3 doesn't incorporate the changes from Listing 3.1, but see Listing 3.9.)

Listing 3.3: A test for the Contact page. **RED**
test/controllers/static_pages_controller_test.rb

```ruby
require 'test_helper'

class StaticPagesControllerTest < ActionController::TestCase

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

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

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

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

At this point, the tests in Listing 3.3 should be **RED**:

Listing 3.4: **RED**

```
$ bundle exec rake test
```

The application code parallels the addition of the About page: first we update the routes (Listing 3.5), then we add a `contact` action to the Static Pages controller (Listing 3.6), and finally we create a Contact view (Listing 3.7). For the last of these, recall the `touch` trick that can be used to create a new file:

```
$ touch app/views/static_pages/contact.html.erb
```

Listing 3.5: Adding a route for the Contact page. **RED**
config/routes.rb

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

Listing 3.6: Adding an action for the Contact page. **RED**
app/controllers/static_pages_controller.rb

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

Listing 3.7: The view for the Contact page. **GREEN**
app/views/static_pages/contact.html.erb

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

We these additions, the tests should be **GREEN**:

Listing 3.8: **GREEN**

```
$ bundle exec rake test
```

Now that we have a **GREEN** test suite, we can refactor the tests[1] in Listing 3.3 to use the base title from Listing 3.1, as seen in Listing 3.9.

Listing 3.9: Refactoring the tests to use a common base title. **GREEN**
test/controllers/static_pages_controller_test.rb

```ruby
require 'test_helper'

class StaticPagesControllerTest < ActionController::TestCase
```

[1]Note that this is a sort of "inverse" refactoring, in that we are refactoring the *tests*, not the application code. Indeed, in this context the *application* code effectively serves as "tests" for the tests!

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

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

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

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

  test "should get contact" do
    get :contact
    assert_response :success
    assert_select "title", "Contact | #{@base_title}"
  end
end
```

After the refactoring in Listing 3.9, the test suite should still be **GREEN**:

Listing 3.10: GREEN

```
$ bundle exec rake test
```

Chapter 4

Solutions to Chapter 4 exercises

4.1 Exercise 1

The solution involves *chaining* the `split`, `shuffle`, and `join` methods, as shown in Listing 4.1. (Because the `shuffle` method uses a random-number generator, your results on the last line of Listing 4.1 may vary.)[1]

Listing 4.1: A string shuffle function.

```
>> def string_shuffle(s)
>>   s.split('').shuffle.join
>> end
>> string_shuffle("foobar")
=> "oobfra"
```

[1]The number of different shuffles of the letters in "foobar" is equal to 6! (the number of permutations of six letters) divided by 2! (the number of permutations of two letters, to account for the double "o" in "foobar"): $N = \frac{6!}{2!} = 6 \times 5 \times 4 \times 3 = 360$. The probability that you'll get the same result shown in the last line of Listing 4.1 is thus $p = \frac{1}{N} = \frac{1}{360} \approx 0.278\%$.

4.2 Exercise 2

Applying the method chaining from Listing 4.1 yields the `shuffle` method shown in Listing 4.2.

Listing 4.2: Adding a `shuffle` method to the `String` class.

```
>> class String
>>   def shuffle
>>     self.split('').shuffle.join
>>   end
>> end
>> "foobar".shuffle
=> "borafo"
```

Inside the `String` class, the use of `self` is optional, so we can even write the `String#shuffle` method as

```
split('').shuffle.join
```

as shown in Listing 4.3.

Listing 4.3: Omitting `self` in the `shuffle` method.

```
>> class String
>>   def shuffle
>>     split('').shuffle.join
>>   end
>> end
>> "foobar".shuffle
=> "bfooar"
```

4.3 Exercise 3

The following Rails console session shows how to create the relevant variables:

```
$ rails console
>> person1 = { first: "Ned", last: "Stark" }
=> {:first=>"Ned", :last => "Stark"}
>> person2 = { first: "Catelyn", last: "Stark" }
=> {:first=>"Catelyn", :last=>"Stark"}
>> person3 = { first: "Arya", last: "Stark" }
=> {:first=>"Arya", :last=>"Stark"}
>> params = { father: person1, mother: person2, child: person3 }
=> {:father=>{:first=>"Ned", :last=>"Stark"}, :mother=>{:first=>"Catelyn",
:last=>"Stark"}, :child=>{:first=>"Arya", :last=>"Stark"}}
>> params[:father]
=> {:first=>"Ned", :last=>"Stark"}
>> params[:mother]
=> {:first=>"Catelyn", :last=>"Stark"}
>> params[:child]
=> {:first=>"Arya", :last=>"Stark"}
>> params[:father][:first]
=> "Ned"
```

4.4 Exercise 4

From the Ruby API entry on Hash, we learn that the `merge` method "[r]eturns a new hash containing the contents of `other_hash` and the contents of `hsh` [where] the value for entries with duplicate keys will be that of `other_hash`." In other words, `merge` combines two hashes, using the values in the *second* hash for any duplicate keys. In the case of merging the hash `{ "a" => 100, "b" => 200 }` with the hash `{ "b" => 300 }`, the value for the key `"a"` comes from the first hash, while the value for the duplicate key `"b"` comes from the second hash:

```
>> { "a" => 100, "b" => 200 }.merge({ "b" => 300 })
=> {"a"=>100, "b"=>300}
```

Chapter 5

Solutions to Chapter 5 exercises

5.1 Exercise 1

We begin with the CSS for the site footer:

```css
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;
}
```

Note that the `footer` tag is duplicated in every rule:

```
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;
}
```

Using SCSS, we can eliminate this duplication via nesting:

```
footer {
  margin-top: 45px;
  padding-top: 5px;
  border-top: 1px solid #eaeaea;
  color: #777;
  a {
    color: #555;
  }
  a:hover {
    color: #222;
  }
  small {
    float: left;
  }
  ul {
    float: right;
    list-style: none;
  }
  ul li {
    float: left;
    margin-left: 15px;
  }
}
```

Note the repetition of the unordered list tag **ul**:

```
ul {
  float: right;
  list-style: none;
}
ul li {
  float: left;
  margin-left: 15px;
}
```

As with the **footer** tag, this duplication can be eliminated using simple nesting:

```
ul {
  float: right;
  list-style: none;
  li {
    float: left;
    margin-left: 15px;
  }
}
```

Consider now the rules for the anchor tag **a**:

```
a {
  color: #555;
}
a:hover {
  color: #222;
}
```

The `a` is duplicated, but the duplication can't be eliminated using simple nesting because of the use of `hover` on the second rule. The solution is to use an ampersand `&` to refer to the parent tag inside the nesting:

```
a {
  color: #555;
  &:hover {
    color: #222;
  }
}
```

Putting these elements together gives the completed footer SCSS shown in Listing 5.1.

Listing 5.1: The completed footer SCSS.

```
footer {
  margin-top: 45px;
  padding-top: 5px;
  border-top: 1px solid #eaeaea;
  color: #777;
  a {
    color: #555;
    &:hover {
      color: #222;
    }
  }
  small {
    float: left;
  }
  ul {
    float: right;
    list-style: none;
    li {
      float: left;
      margin-left: 15px;
    }
  }
}
```

Finally, we replace the hard-coded colors in Listing 5.1 with named variables:

```
#eaeaea -> $gray-medium-light
#777    -> $gray-light
#555    -> $gray
#222    -> $gray-darker
```

This yields the final result, shown in Listing 5.2.

Listing 5.2: The final footer SCSS.

```scss
footer {
  margin-top: 45px;
  padding-top: 5px;
  border-top: 1px solid $gray-medium-light;
  color: $gray-light;
  a {
    color: $gray;
    &:hover {
      color: $gray-darker;
    }
  }
  small {
    float: left;
  }
  ul {
    float: right;
    list-style: none;
    li {
      float: left;
      margin-left: 15px;
    }
  }
}
```

5.2 Exercise 2

We'll think of the integration test as a simulation of a user clicking around.[1] In this context, we can visit the signup page as follows:

```
get signup_path
```

We can then test for the correct page title using `assert_select`:

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

Adding these steps to the existing test results in Listing 5.3.

Listing 5.3: Testing the signup page's title. **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 signup_path
    assert_select "title", "Sign up | Ruby on Rails Tutorial Sample App"
  end
end
```

At this point, the test suite should be **GREEN**:

[1]This isn't quite how integration tests work; in particular, there's no direct analogue to issuing a click. This limitation can be lifted by the Capybara library, which introduces convenient syntax like `click_link "Sign up"`. Unfortunately, Capybara doesn't work well with the authentication system developed later in the book, and it also adds significant syntactic complexity, so Capybara has been omitted from the tutorial in favor of the "default stack" integration tests.

Listing 5.4: GREEN

```
$ bundle exec rake test
```

5.3 Exercise 3

To make a working test, we first need to create the corresponding test file:

```
$ touch test/helpers/application_helper_test.rb
```

Then we fill in the first `FILL_IN` with the base title, and fill in the second `FILL_IN` with a combination of the page title and base title. The result appears in Listing 5.5.

Listing 5.5: A direct test 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,         "Ruby on Rails Tutorial Sample App"
    assert_equal full_title("Help"), "Help | Ruby on Rails Tutorial Sample App"
  end
end
```

With the direct test of the `full_title` helper in Listing 5.5, we are now in a position to put the helper to use in our tests if we like. The way to do this is first to *include* the Application helper into the test helper, as shown in Listing 5.6.

Listing 5.6: Including the Application helper in tests.
test/test_helper.rb

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

We can then update code like Listing 5.3 to use the `full_title` helper, as shown in Listing 5.7.

Listing 5.7: 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 signup_path
    assert_select "title", full_title("Sign up")
  end
end
```

In this context, it's important to have the direct test in Listing 5.5 because otherwise a typo in the title could go undetected. In other words, if we replaced all of the title tests with uses of `full_title`, they would still pass even if we misspelled "Tutorial" as "Tutoial", as shown in Listing 5.8.

Listing 5.8: Misspelling “Tutorial” as “Tutoial” in the title.
app/helpers/application_helper.rb

```ruby
module ApplicationHelper

  # Returns the full title on a per-page basis.
  def full_title(page_title = '')
    base_title = "Ruby on Rails Tutoial Sample App"
    if page_title.empty?
      base_title
    else
      "#{page_title} | #{base_title}"
    end
  end
end
```

With the mistake in Listing 5.8, the test in Listing 5.7 would still be **GREEN**, but the test in Listing 5.5 would be **RED** and would therefore catch the error.

Upon reverting the error introduced in Listing 5.8, the test suite should be **GREEN**:

Listing 5.9: GREEN

```
$ bundle exec rake test
```

Chapter 6

Solutions to Chapter 6 exercises

6.1 Exercise 1

The test in Listing 6.1 is given as part of the exercise. It should initally be **GREEN**.

Listing 6.1: A test for email downcasing. **GREEN**
test/models/user_test.rb

```
require 'test_helper'

class UserTest < ActiveSupport::TestCase

  def setup
    @user = User.new(name: "Example User", email: "user@example.com",
                     password: "foobar", password_confirmation: "foobar")
  end
  .
  .
  .
  test "email addresses should be unique" do
    duplicate_user = @user.dup
    duplicate_user.email = @user.email.upcase
    @user.save
    assert_not duplicate_user.valid?
  end
  test "email addresses should be saved as lower-case" do
    mixed_case_email = "Foo@ExAMPle.CoM"
    @user.email = mixed_case_email
    @user.save
    assert_equal mixed_case_email.downcase, @user.reload.email
  end

  test "password should have a minimum length" do
    @user.password = @user.password_confirmation = "a" * 5
    assert_not @user.valid?
  end
end
```

To verify that the test in Listing 6.1 tests the right thing, we comment out the `before_save` callback in the User model, as shown in Listing 6.2.

Listing 6.2: Commenting out the `before_save` callback. **RED**
app/models/user.rb

```
class User < ActiveRecord::Base
  # before_save { self.email = email.downcase }
  validates :name, presence: true, length: { maximum: 50 }
  VALID_EMAIL_REGEX = /\A[\w+\-.]+@[a-z\d\-.]+\.[a-z]+\z/i
  validates :email, presence: true, length: { maximum: 255 },
                    format: { with: VALID_EMAIL_REGEX },
                    uniqueness: { case_sensitive: false }
  has_secure_password
  validates :password, length: { minimum: 6 }
end
```

The test suite should now be **RED**:

Listing 6.3: **RED**

```
$ bundle exec rake test
```

We now uncomment the callback to get back to **GREEN** (Listing 6.4).

Listing 6.4: Uncommenting the `before_save` callback. **GREEN**
app/models/user.rb

```
class User < ActiveRecord::Base
  before_save { self.email = email.downcase }
  validates :name, presence: true, length: { maximum: 50 }
  VALID_EMAIL_REGEX = /\A[\w+\-.]+@[a-z\d\-.]+\.[a-z]+\z/i
  validates :email, presence: true, length: { maximum: 255 },
                    format: { with: VALID_EMAIL_REGEX },
                    uniqueness: { case_sensitive: false }
  has_secure_password
  validates :password, length: { minimum: 6 }
end
```

The test suite should now be **GREEN**:

Listing 6.5: GREEN

```
$ bundle exec rake test
```

6.2 Exercise 2

As is often the case in Ruby, the use of the "bang" `!` indicates that the method in question *mutates* the given variable:

```
>> email = "MHARTL@EXAMPLE.COM"
=> "MHARTL@EXAMPLE.COM"
>> email.downcase
=> "mhartl@example.com"
>> email
=> "MHARTL@EXAMPLE.COM"
>> email.downcase!
=> "mhartl@example.com"
>> email
=> "mhartl@example.com"
```

The final two lines show that, unlike `downcase`, the `downcase!` method modifies the `email` variable itself.

Applying this idea to the User model's `before_save` callback gives the implementation in Listing 6.6.

Listing 6.6: An alternate implementation of the `before_save` callback. **GREEN**
app/models/user.rb

```
class User < ActiveRecord::Base
  before_save { email.downcase! }
  validates :name, presence: true, length: { maximum: 50 }
  VALID_EMAIL_REGEX = /\A[\w+\-.]+@[a-z\d\-.]+\.[a-z]+\z/i
  validates :email, presence: true, length: { maximum: 255 },
                    format: { with: VALID_EMAIL_REGEX },
                    uniqueness: { case_sensitive: false }
  has_secure_password
  validates :password, length: { minimum: 6 }
end
```

We can check that Listing 6.6 works by verifying that the test suite is still **GREEN**:

Listing 6.7: **GREEN**

```
$ bundle exec rake test
```

6.3 Exercise 3

We start by adding *foo@bar..com* to Listing 6.8 as suggested.

Listing 6.8: Tests for email format validation. **RED**
test/models/user_test.rb

```
require 'test_helper'

class UserTest < ActiveSupport::TestCase

  def setup
    @user = User.new(name: "Example User", email: "user@example.com")
  end
  .
  .
  .
  test "email validation should reject invalid addresses" do
    invalid_addresses = %w[user@example,com user_at_foo.org user.name@example.
                           foo@bar_baz.com foo@bar+baz.com
                           foo@bar..com]
    invalid_addresses.each do |invalid_address|
      @user.email = invalid_address
      assert_not @user.valid?
    end
  end
  .
  .
  .
end
```

The test suite should now be **RED**:

Listing 6.9: RED

```
$ bundle exec rake test
```

To get the test to **GREEN**, we replace

```
[a-z\d\-.]+
```

with

```
[a-z\d\-]+(\.[a-z\d\-]+)*
```

in the User model's valid email regex. The former matches one or more repeated elements containing letters, digits, hyphens, or dots, which is why it allows double dots in email domain names. The second expression, in contrast, matches one or more repeated expressions containing letters, digits, and hyphens, but *not* dots, followed by *zero* or more expressions of *exactly one* dot and one or more expressions containing letters, digits, or hyphens. As a result, the second regular expression *won't* match double dots in the domain name.[1]

[1]Regular expressions can be tricky, so if this discussion is too abstract I recommend experimenting interactively with Rubular.

Incorporating the results of the above discussion into the User model gives the code shown in Listing 6.10.

Listing 6.10: Disallowing double dots in email domain names. **GREEN**
app/models/user.rb

```
class User < ActiveRecord::Base
  before_save { email.downcase! }
  validates :name, presence: true, length: { maximum: 50 }
  VALID_EMAIL_REGEX = /\A[\w+\-.]+@[a-z\d\-]+(\.[a-z\d\-]+)*\.[a-z]+\z/i
  validates :email, presence: true, length: { maximum: 255 },
                    format: { with: VALID_EMAIL_REGEX },
                    uniqueness: { case_sensitive: false }
  has_secure_password
  validates :password, length: { minimum: 6 }
end
```

The test suite should now be **GREEN** as required:

Listing 6.11: **GREEN**

```
$ bundle exec rake test
```

Chapter 7

Solutions to Chapter 7 exercises

7.1 Exercise 1

The improved `gravatar_for` code is given in the exercise, as shown in Listing 7.1.

Listing 7.1: Adding an options hash in the `gravatar_for` helper.
app/helpers/users_helper.rb

```
module UsersHelper

  # Returns the Gravatar for the given user.
  def gravatar_for(user, options = { size: 80 })
    gravatar_id = Digest::MD5::hexdigest(user.email.downcase)
    size = options[:size]
    gravatar_url = "https://secure.gravatar.com/avatar/#{gravatar_id}?s=#{size}"
    image_tag(gravatar_url, alt: user.name, class: "gravatar")
  end
end
```

The definition of `gravatar_for` allows for code like this:

```
gravatar_for @user, size: 50      # Returns a 50x50 Gravatar
```

This arranges for a 50x50 Gravatar image. Because of the `options` hash

```
options = { size: 80 }
```

the default size is 80x80, which is what you get if you omit `size`:

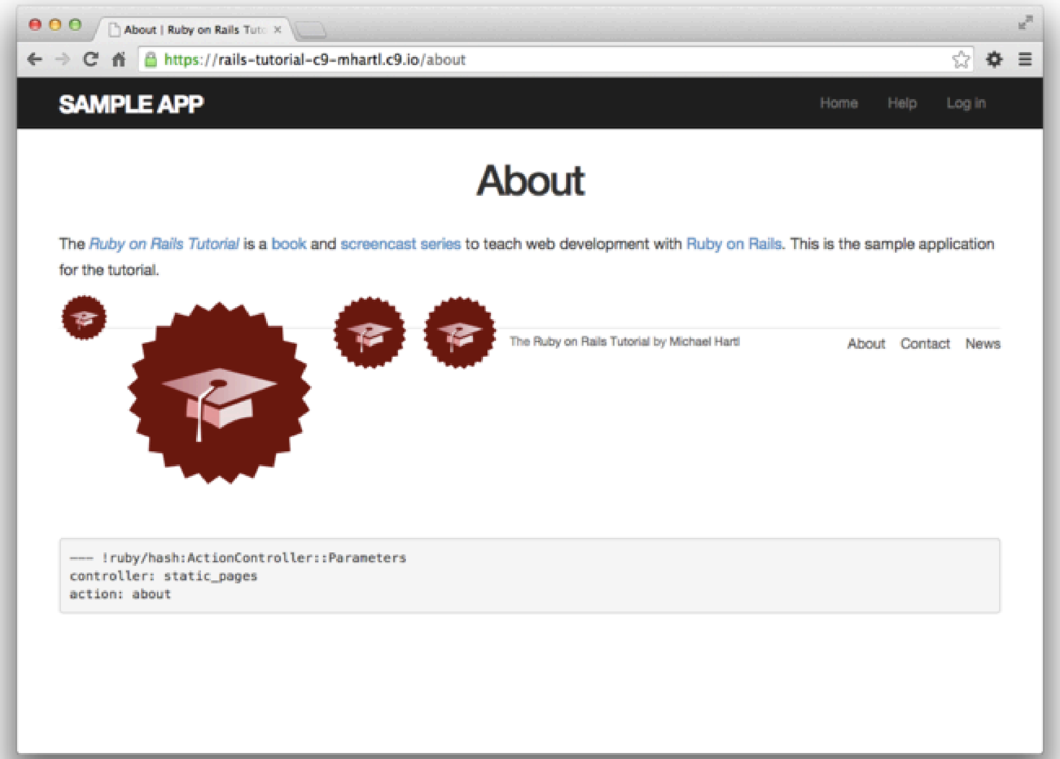
```
gravatar_for @user                # Returns an 80x80 Gravatar
```

As an illustration, we'll add some extra calls to `gravatar_for` to the About page, as shown in Listing 7.2.

Listing 7.2: Adding some Gravatars to the About page for purposes of illustration.
app/views/static_pages/about.html.erb

```
<% provide(:title, "About") %>
<h1>About</h1>
<p>
  The <a href="http://www.railstutorial.org/"><em>Ruby on Rails
  Tutorial</em></a> is a
  <a href="http://www.railstutorial.org/book">book</a> and
  <a href="http://screencasts.railstutorial.org/">screencast series</a>
  to teach web development with
  <a href="http://rubyonrails.org/">Ruby on Rails</a>.
  This is the sample application for the tutorial.
</p>

<%= gravatar_for User.first, size: 50 %>
<%= gravatar_for User.first, size: 200 %>
<%= gravatar_for User.first, size: 80 %>
<%= gravatar_for User.first %>
```

The result appears in Figure 7.1. Note that, as promised, the 80x80 Gravatar and the default Gravatar are the same size.

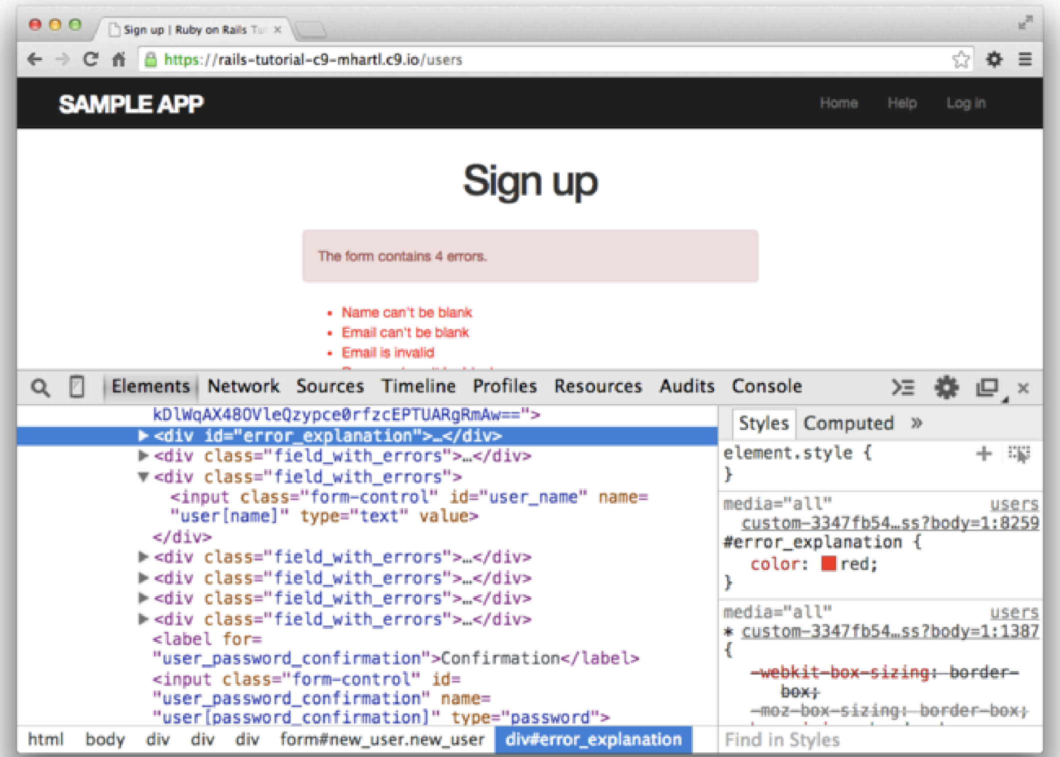

About | Ruby on Rails Tuto
https://rails-tutorial-c9-mhartl.c9.io/about
SAMPLE APP
Home
Help
Log in
The Ruby on Rails Tutorial by Michael Hartl
About
Contact
News

About

The *Ruby on Rails Tutorial* is a book and screencast series to teach web development with Ruby on Rails. This is the sample application for the tutorial.

```
--- !ruby/hash:ActionController::Parameters
controller: static_pages
action: about
```

Figure 7.1: Gravatars of various sizes on the About page.



Sign up

The form contains 4 errors.

- Name can't be blank
- Email can't be blank
- Email is invalid

```
kDlWqAX48OVleQzypce0rfzcEPTUARgRmAw==">
▶<div id="error_explanation">…</div>
▶<div class="field_with_errors">…</div>
▼<div class="field_with_errors">
   <input class="form-control" id="user_name" name=
   "user[name]" type="text" value>
 </div>
▶<div class="field_with_errors">…</div>
▶<div class="field_with_errors">…</div>
▶<div class="field_with_errors">…</div>
▶<div class="field_with_errors">…</div>
 <label for=
 "user_password_confirmation">Confirmation</label>
 <input class="form-control" id=
 "user_password_confirmation" name=
 "user[password_confirmation]" type="password">
```

```
html  body  div  div  div  form#new_user.new_user  div#error_explanation
```

```
element.style {
}
media="all"                                   users
  custom-3347fb54…ss?body=1:8259
#error_explanation {
  color: 🟥red;
}
media="all"                                   users
* custom-3347fb54…ss?body=1:1387
{
  ~~-webkit-box-sizing: border-
    box;~~
  ~~-moz-box-sizing: border-box;~~
```

Figure 7.2: *Inspecting the error messages.*

7.2 Exercise 2

As seen in Figure 7.2, the error messages consist of **`div`** tags with CSS id **`error_explanation`** and CSS class **`field_with_errors`**. Inside of a CSS file, such elements would be accessed with a hash **`#`** and dot **`.`**, respectively:

```
div#error_explanation { ... }
div.field_with_errors { ... }
```

The **`assert_select`** method understands this syntax, yielding the test shown in Listing 7.3.

Listing 7.3: A test of the error messages. **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, 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
  .
  .
  .
end
```

With the test in Listing 7.3, the test suite should still be **GREEN**:

Listing 7.4: **GREEN**

```
$ bundle exec rake test
```

7.3 Exercise 3

As noted in the exercise, even testing for the right key, much less the text, is likely to be brittle. For example, a test for the `:danger` key would fail if we changed it to `:warning`, but that's not necessarily the kind of change we would want to have break our test suite. In circumstances such as this one, I prefer to test only that the flash isn't `nil`, which should be true regardless of the other details of the implementation. This gives the test shown in Listing 7.5.

Listing 7.5: A test of the flash. **GREEN**
test/integration/users_signup_test.rb

```ruby
require 'test_helper'
  .
  .
  .
  test "valid signup information" do
    get signup_path
    assert_difference 'User.count', 1 do
      post_via_redirect users_path, user: { name:  "Example User",
                                            email: "user@example.com",
                                            password:              "password",
                                            password_confirmation: "password" }
    end
    assert_template 'users/show'
    assert_not flash.nil?
  end
end
```

It's worth noting that MiniTest includes a method called `assert_not_nil`, so Listing 7.5 can actually be rewritten as shown in Listing 7.6.

Listing 7.6: Using `assert_not_nil` to test the flash. **GREEN**
test/integration/users_signup_test.rb

```ruby
require 'test_helper'
  .
  .
  .
  test "valid signup information" do
    get signup_path
    assert_difference 'User.count', 1 do
      post_via_redirect users_path, user: { name:  "Example User",
                                            email: "user@example.com",
                                            password:              "password",
                                            password_confirmation: "password" }
    end
    assert_template 'users/show'
    assert_not_nil flash
  end
end
```

Whether you use Listing 7.5 or Listing 7.6, the test suite should be **GREEN**:

Listing 7.7: GREEN

```
$ bundle exec rake test
```

7.4 Exercise 4

The versatile `content_tag` helper takes a tag name (typically as a symbol), the tag content, and an optional hash, and returns the relevant HTML tag:

```
content_tag(:span, "foobar")
# => <span>foobar</span>
```

The `content_tag` helper's options hash allows the inclusion of CSS ids, classes, etc.:

```
content_tag(:div, "Oops!", class: "alert alert-danger")
# => <div class="alert alert-danger">Oops!</div>
```

Using the ideas described above, we can replace hard-to-parse code like

```
<div class="alert alert-<%= message_type %>"><%= message %></div>
```

with friendlier code like this:

```
<%= content_tag(:div, message, class: "alert alert-#{message_type}") %>
```

Applying this to the site layout gives the flash message shown in Listing 7.8.

Listing 7.8: The `flash` ERb in the site layout using `content_tag`.
app/views/layouts/application.html.erb

```
<!DOCTYPE html>
<html>
      .
      .
      .
      <% flash.each do |message_type, message| %>
        <%= content_tag(:div, message, class: "alert alert-#{message_type}") %>
      <% end %>
      .
      .
      .
</html>
```

Chapter 8

Solutions to Chapter 8 exercises

8.1 Exercise 1

Changing the class methods `User.new_token` and `User.digest` to use `self` in place of `User` gives Listing 8.1.

Listing 8.1: Defining the new token and digest methods using `self`. GREEN
app/models/user.rb

```
class User < ActiveRecord::Base
  .
  .
  .
  # Returns the hash digest of the given string.
  def self.digest(string)
    cost = ActiveModel::SecurePassword.min_cost ? BCrypt::Engine::MIN_COST :
                                                  BCrypt::Engine.cost
    BCrypt::Password.create(string, cost: cost)
  end

  # Returns a random token.
  def self.new_token
    SecureRandom.urlsafe_base64
  end
  .
  .
  .
end
```

The tests should still be **GREEN**:

Listing 8.2: GREEN

```
$ bundle exec rake test
```

Placing the class methods inside the commonly used but confusing `class «  self` block gives Listing 8.3.

Listing 8.3: Defining the new token and digest methods using `class << self`. **GREEN**

app/models/user.rb

```
class User < ActiveRecord::Base
  .
  .
  .
  class << self
    # Returns the hash digest of the given string.
    def digest(string)
      cost = ActiveModel::SecurePassword.min_cost ? BCrypt::Engine::MIN_COST :
                                                    BCrypt::Engine.cost
      BCrypt::Password.create(string, cost: cost)
    end

    # Returns a random token.
    def new_token
      SecureRandom.urlsafe_base64
    end
  end
  .
  .
  .
```

As before, the tests should still be **GREEN**:

Listing 8.4: GREEN

```
$ bundle exec rake test
```

8.2 Exercise 2

In order to use `assigns` to access the user object inside tests, we need to change `user` to `@user` in the Sessions controller's `create` action. The template code for an updated `create` action appears in Listing 8.6; to convert `user` to an instance varible, we need to change the question marks in Listing 8.6 to @ signs, as shown in Listing 8.5.

Listing 8.5: A template for using an instance variable in the `create` action.
app/controllers/sessions_controller.rb

```
class SessionsController < ApplicationController

  def new
  end

  def create
    ?user = User.find_by(email: params[:session][:email].downcase)
    if ?user && ?user.authenticate(params[:session][:password])
      log_in ?user
      params[:session][:remember_me] == '1' ? remember(?user) : forget(?user)
      redirect_to ?user
    else
      flash.now[:danger] = 'Invalid email/password combination'
      render 'new'
    end
  end

  def destroy
    log_out if logged_in?
    redirect_to root_url
  end
end
```

Listing 8.6: Changing `?` to `@` to define an instance variable `@user`.
app/controllers/sessions_controller.rb

```
class SessionsController < ApplicationController

  def new
  end

  def create
    @user = User.find_by(email: params[:session][:email].downcase)
    if @user && @user.authenticate(params[:session][:password])
      log_in @user
      params[:session][:remember_me] == '1' ? remember(@user) : forget(@user)
      redirect_to @user
    else
      flash.now[:danger] = 'Invalid email/password combination'
      render 'new'
    end
  end

  def destroy
    log_out if logged_in?
    redirect_to root_url
  end
end
```

With the code as in Listing 8.6, we can verify directly that the `cookies` remember token has the right value, as shown in Listing 8.7.

Listing 8.7: An improved "remember me" test. GREEN
test/integration/users_login_test.rb

```
require 'test_helper'

class UsersLoginTest < ActionDispatch::IntegrationTest

  def setup
    @user = users(:michael)
  end
  .
  .
  .
  test "login with remembering" do
    log_in_as(@user, remember_me: '1')
    assert_equal assigns(:user).remember_token, cookies['remember_token']
  end
  test "login without remembering" do
    log_in_as(@user, remember_me: '0')
    assert_nil cookies['remember_token']
  end
  .
  .
  .
end
```

With the code in Listing 8.7, the tests should still be **GREEN**:

Listing 8.8: GREEN

```
$ bundle exec rake test
```

Chapter 9

Solutions to Chapter 9 exercises

9.1 Exercise 1

The forwarding location is stored in `session[:forwarding_url]`, so we simply need to assert that its value is `nil` after a successful redirect. To make sure it's testing the right thing, we've also added an assertion that the forwarding URL *isn't* `nil` after hitting the protected page. (There's no need to check for the right value, as that's already tested by `assert_redirected_to @user`.) The result appears in Listing 9.1.

Listing 9.1: A test for forwarding only once. GREEN
test/integration/users_edit_test.rb

```
require 'test_helper'

class UsersEditTest < ActionDispatch::IntegrationTest

  def setup
    @user = users(:michael)
  end
  .
  .
  .
  test "successful edit with friendly forwarding" do
```

```
    get edit_user_path(@user)
    assert_not_nil session[:forwarding_url]
    log_in_as(@user)
    assert_redirected_to edit_user_path(@user)
    name  = "Foo Bar"
    email = "foo@bar.com"
    patch user_path(@user), user: { name:  name,
                                    email: email,
                                    password:              "foobar",
                                    password_confirmation: "foobar" }
    assert_not flash.empty?
    assert_redirected_to @user
    assert_nil session[:forwarding_url]
    @user.reload
    assert_equal @user.name,  name
    assert_equal @user.email, email
  end
end
```

At this point, the tests should still be **GREEN**:

Listing 9.2: **GREEN**

```
$ bundle exec rake test
```

9.2 Exercise 2

The layout links test currently appears as in Listing 9.3.

Listing 9.3: 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
```

Using the **`log_in_as`** helper, we can test for the changes in the layout links as shown in Listing 9.4.

Listing 9.4: Adding layout links tests for logged-in users. **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
    assert_select "a[href=?]", login_path
    user = users(:michael)
    log_in_as(user)
    get root_path
    assert_select "a[href=?]", logout_path
    assert_select "a[href=?]", users_path
    assert_select "a[href=?]", user_path(user)
    assert_select "a[href=?]", edit_user_path(user)
  end
end
```

Note that Listing 9.4 also adds an assertion for the correct login link for non-logged-in users.

At this point, the tests should still be **GREEN**:

Listing 9.5: **GREEN**

```
$ bundle exec rake test
```

9.3 Exercise 3

To make sure we're properly testing access to the **admin** attribute, we'll first add it to the list of permitted parameters, as shown in Listing 9.6.

Listing 9.6: Making **admin** editable through the Web. **GREEN**
app/controllers/users_controller.rb

```
class UsersController < ApplicationController
  .
  .
  .
  private

    def user_params
      params.require(:user).permit(:name, :email, :password,
                                   :password_confirmation,
                                   :admin)
    end
end
```

Despite this massive security hole, our test suite is still **GREEN**:

Listing 9.7: **GREEN**

```
$ bundle exec rake test
```

To catch this error, we'll add a failing test, as shown in Listing 9.8.

Listing 9.8: Testing that the **admin** attribute is forbidden. **RED**
test/controllers/users_controller_test.rb

```
require 'test_helper'

class UsersControllerTest < ActionController::TestCase

  def setup
    @user       = users(:michael)
    @other_user = users(:archer)
  end
  .
  .
  .
  test "should redirect update when logged in as wrong user" do
    log_in_as(@other_user)
    patch :update, id: @user, user: { name: @user.name, email: @user.email }
    assert_redirected_to root_url
  end

  test "should not allow the admin attribute to be edited via the web" do
    log_in_as(@other_user)
    assert_not @other_user.admin?
    patch :update, id: @other_user, user: { password:              'password',
                                            password_confirmation: 'password',
                                            admin: true }
    assert_not @other_user.reload.admin?
  end
  .
  .
  .
end
```

Note the use of **`reload`** in Listing 9.8 to pull the other user's information out of the database:

```
@other_user.reload.admin?
```

If instead we wrote

```
@other_user.admin?
```

then the test suite would pass *no matter what*, even with the insecure code from Listing 9.6. Thus, such a test would give us a dangerous false sense of security, appearing to protect against making the **`admin`** attribute editable through the web while in fact doing no such thing. And yet, it would be incredibly easy to forget to use **`reload`** in this context, which is why it's so important to get to **RED** before getting back to **GREEN**.

As required, the test suite should now be **RED**:

Listing 9.9: RED

```
$ bundle exec rake test
```

To get back to GREEN, we just remove the `:admin` attribute from Listing 9.6, as shown in Listing 9.10.

Listing 9.10: Removing `:admin` from the permitted parameters. GREEN
app/controllers/users_controller.rb

```
class UsersController < ApplicationController
  .
  .
  .
  private

    def user_params
      params.require(:user).permit(:name, :email, :password,
                                   :password_confirmation)
    end
end
```

The tests should now be GREEN:

Listing 9.11: GREEN

```
$ bundle exec rake test
```

Having gone through the full Red–Green cycle, we can be confident that our test suite will catch any regressions if the `admin` attribute is accidentally exposed to outside attack.

9.4 Exercise 4

We first need to create the relevant partial:

```
$ touch app/views/users/_fields.html.erb
```

We next fill it with the contents of Listing 9.12.

Listing 9.12: A partial for the `new` and `edit` form fields.
app/views/users/_fields.html.erb

```
<%= render 'shared/error_messages' %>

<%= f.label :name %>
<%= f.text_field :name, class: 'form-control' %>

<%= f.label :email %>
<%= f.email_field :email, class: 'form-control' %>

<%= f.label :password %>
<%= f.password_field :password, class: 'form-control' %>

<%= f.label :password_confirmation, "Confirmation" %>
<%= f.password_field :password_confirmation, class: 'form-control' %>
```

Because Listing 9.12 uses the form variable `f`, we need to pass this variable to the partial, which we can do when rendering the partial as follows:

```
<%= render 'fields', f: f %>
```

Note that the two occurrences of `f` here are different; if Listing 9.12 used `dude` in place of `f`, we would write

```
<%= render 'fields', dude: f %>
```

instead.

Rendering the partial as described above in both the `new` and `edit` templates gives the updated views shown in Listing 9.13 and Listing 9.14.

Listing 9.13: The signup view with a partial.
app/views/users/new.html.erb

```
<% provide(:title, 'Sign up') %>
<h1>Sign up</h1>

<div class="row">
  <div class="col-md-6 col-md-offset-3">
    <%= form_for(@user) do |f| %>
      <%= render 'fields', f: f %>
      <%= f.submit "Create my account", class: "btn btn-primary" %>
    <% end %>
  </div>
</div>
```

Listing 9.14: The user edit view with a partial.
app/views/users/edit.html.erb

```
<% provide(:title, "Edit user") %>
<h1>Update your profile</h1>

<div class="row">
  <div class="col-md-6 col-md-offset-3">
    <%= form_for(@user) do |f| %>
      <%= render 'fields', f: f %>
      <%= f.submit "Save changes", class: "btn btn-primary" %>
    <% end %>

    <div class="gravatar_edit">
      <%= gravatar_for @user %>
      <a href="http://gravatar.com/emails">change</a>
    </div>
  </div>
</div>
```

After a change of this nature, it's always a good idea to verify that the test suite is still **GREEN**:

Listing 9.15: **GREEN**

```
$ bundle exec rake test
```

Chapter 10

Solutions to Chapter 10 exercises

10.1 Exercise 1

To check for some indication of password reset expiration without tying our test to a specific implementation, we simply verify that the word "expires" appears somewhere on the page. The key to this is `response.body`, which (despite its name) contains the full HTML source of the rendered page. The resulting test appears in Listing 10.1.

Listing 10.1: A test for an expired password reset. GREEN
test/integration/password_resets_test.rb

```ruby
require 'test_helper'

class PasswordResetsTest < ActionDispatch::IntegrationTest

  def setup
    ActionMailer::Base.deliveries.clear
    @user = users(:michael)
  end
  .
  .
  .
  test "expired token" do
    # Create a matching reset token/digest pair.
    post password_resets_path, password_reset: { email: @user.email }
    @user = assigns(:user)
    @user.update_attribute(:reset_sent_at, 3.hours.ago)

    # Check that the edit action is protected.
    get edit_password_reset_path(@user.reset_token)
    assert_redirected_to root_url
    assert_match /expired/i, response.body

    # Check that the update action is protected.
    patch_via_redirect password_reset_path(@user.reset_token),
                       email: @user.email,
                       user: { password:              "foobar",
                               password_confirmation: "foobar" }
    assert_redirected_to root_url
    assert_match /expired/i, response.body
  end
end
```

The test suite should be **GREEN**:

Listing 10.2: **GREEN**

```
$ bundle exec rake test
```

10.2 Exercise 2

To select only the activated users in the Users controller `index` action, we use Active Record's `where` method:

```
User.where(activated: true)
```

Meanwhile, in the `show` action, we redirect unless the user is activated. The result appears in Listing 10.3.

Listing 10.3: Code to show only active users.
app/controllers/users_controller.rb

```
class UsersController < ApplicationController
  .
  .
  .
  def index
    @users = User.where(activated: true).paginate(page: params[:page])
  end

  def show
    @user = User.find(params[:id])
    redirect_to root_url and return unless @user.activated?
  end
  .
  .
  .
end
```

To test the code in Listing 10.3, we can add to the test for valid signup by (a) checking that the newly created user doesn't appear on the Users index and (b) checking that the user's profile page redirects properly. (Our test suite already covers both the index and profile pages for activated users, so we don't need to add assertions for these cases.) The result appears in Listing 10.4.

Listing 10.4: Adding account activation to the user signup test. **GREEN**
test/integration/users_signup_test.rb

```
require 'test_helper'

class UsersSignupTest < ActionDispatch::IntegrationTest

  def setup
    ActionMailer::Base.deliveries.clear
  end
  .
  .
  .
  test "valid signup information with account activation" do
    get signup_path
    name  = "Example User"
    email = "user@example.com"
    password = "foobar"
    assert_difference 'User.count', 1 do
      post users_path, user: { name:  name,
                               email: email,
                               password:              password,
                               password_confirmation: password }
    end
    assert_equal ActionMailer::Base.deliveries.size, 1
    user = assigns(:user)
    assert_not user.activated?
    # Try to log in before activation.
    log_in_as(user)
    assert_not is_logged_in?
    # Index page
    get users_path
    assert_no_match user.name, response.body
    # Profile page
    get user_path(user)
    assert_redirected_to root_url
    # Invalid activation token
    get edit_account_activation_path("invalid token")
    assert_not is_logged_in?
    # Valid token, wrong email
    get edit_account_activation_path(user.activation_token, email: 'wrong')
    assert_not is_logged_in?
    # Valid activation token
    get edit_account_activation_path(user.activation_token, email: user.email)
    assert user.reload.activated?
    follow_redirect!
    assert_template 'users/show'
    assert is_logged_in?
  end
end
```

Listing 10.4 checks that the user doesn't appear on the index page by asserting that there is no match between the user's name and the page's HTML:

```
assert_no_match user.name, response.body
```

(Note that, as with the complementary **`assert_match`** method used in Listing 10.1, the first argument can be either a string or a regular expression.) Asserting (no) matches in this way is a robust and flexible method for testing HTML results without tying our test too closely to any particular implementation.

Of course, at this point the test suite should be **GREEN**:

Listing 10.5: GREEN

```
$ bundle exec rake test
```

10.3 Exercise 3

Multiple calls to `update_attribute` can be replaced with a single call to `update_columns`. For example, the `activate` method's code

```
update_attribute(:activated,    true)
update_attribute(:activated_at, Time.zone.now)
```

can be converted to a nearly equivalent call using `update_columns`:

```
update_columns(activated: true, activated_at: Time.zone.now)
```

(The only difference is that the latter hits the database only once insted of twice.) Similarly, the `create_reset_digest` method can be converted from

```
self.activation_token  = User.new_token
self.activation_digest = User.digest(activation_token)
```

to

```
update_columns(reset_digest:  User.digest(reset_token),
               reset_sent_at: Time.zone.now)
```

Putting these changes together yields the updated User model shown in Listing 10.6.

Listing 10.6: Using `update_columns`. GREEN
app/models/user.rb

```
class User < ActiveRecord::Base
  attr_accessor :remember_token, :activation_token, :reset_token
  before_save   :downcase_email
  before_create :create_activation_digest
  .
  .
  .
  # Activates an account.
  def activate
    update_columns(activated: true, activated_at: Time.zone.now)
  end

  # Sends activation email.
  def send_activation_email
    UserMailer.account_activation(self).deliver_now
  end

  # Sets the password reset attributes.
  def create_reset_digest
    self.reset_token = User.new_token
    update_columns(reset_digest:  User.digest(reset_token),
                   reset_sent_at: Time.zone.now)
  end

  # Sends password reset email.
  def send_password_reset_email
    UserMailer.password_reset(self).deliver_now
  end

  private

    # Converts email to all lower-case.
    def downcase_email
      self.email = email.downcase
    end

    # Creates and assigns the activation token and digest.
    def create_activation_digest
      self.activation_token  = User.new_token
      self.activation_digest = User.digest(activation_token)
    end
end
```

Chapter 11

Solutions to Chapter 11 exercises

11.1 Exercise 1

The current Home page template consists of two large pieces, with the displayed view depending on whether the user is logged in or not (Listing 11.1).

Listing 11.1: The (rather messy) view for the Home page.
app/views/static_pages/home.html.erb

```
<% if logged_in? %>
  <div class="row">
    <aside class="col-md-4">
      <section class="user_info">
        <%= render 'shared/user_info' %>
      </section>
      <section class="micropost_form">
        <%= render 'shared/micropost_form' %>
      </section>
    </aside>
    <div class="col-md-8">
      <h3>Micropost Feed</h3>
      <%= render 'shared/feed' %>
    </div>
  </div>
<% else %>
  <div class="center jumbotron">
    <h1>Welcome to the Sample App</h1>
    <h2>
      This is the home page for the
      <a href="http://www.railstutorial.org/">Ruby on Rails Tutorial</a>
      sample application.
    </h2>

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

  <%= link_to image_tag("rails.png", alt: "Rails logo"),
              'http://rubyonrails.org/' %>

<% end %>
```

To make Listing 11.1 more manageable, we'll create a partial for each of the pieces:

```
$ touch app/views/static_pages/_logged_in_home.html.erb
$ touch app/views/static_pages/_non_logged_in_home.html.erb
```

The two branches in Listing 11.1 then go in their respective partials, as shown in Listing 11.2 and Listing 11.3.

Listing 11.2: A Home page partial for logged-in users.
app/views/static_pages/_logged_in_home.html.erb

```
<div class="row">
  <aside class="col-md-4">
    <section class="user_info">
      <%= render 'shared/user_info' %>
    </section>
    <section class="micropost_form">
      <%= render 'shared/micropost_form' %>
    </section>
  </aside>
  <div class="col-md-8">
    <h3>Micropost Feed</h3>
    <%= render 'shared/feed' %>
  </div>
</div>
```

Listing 11.3: A Home page partial for non-logged-in users.
app/views/static_pages/_non_logged_in_home.html.erb

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

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

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

<%= link_to image_tag("rails.png", alt: "Rails logo"),
            'http://rubyonrails.org/' %>
```

With the partials in Listing 11.2 and Listing 11.3, the Home page view is simplified dramatically (Listing 11.4).

Listing 11.4: A much-simplified view for the Home page. **RED**
app/views/static_pages/home.html.erb

```
<% if logged_in? %>
  <%= render 'logged_in_home' %>
<% else %>
  <%= render 'non_logged_in_home' %>
<% end %>
```

Running the test suite produces a surprise:

Listing 11.5: **RED**

```
$ bundle exec rake test
ERROR["test_micropost_interface", MicropostInterfaceTest, 10.091842544]
 test_micropost_interface#MicropostInterfaceTest (10.09s)
ActionView::Template::Error
  test/integration/microposts_interface_test.rb:14
```

The test suite, which we might have expected to be **GREEN**, in fact is **RED**. By inspecting the error message (whose exact line numbers may vary), we find that the error occurs in the micropost interface test when submitting invalid micropost information. The reason for the failure is that the Home page for logged-in users is being rendered from the *Microposts* controller, but the call to **`render`** in Listing 11.4 only works inside the Static Pages controller. We can fix this by explicitly including the controller name in the call to **`render`**, as shown in Listing 11.6.

Listing 11.6: A corrected view for the Home page. **GREEN**
app/views/static_pages/home.html.erb

```
<% if logged_in? %>
  <%= render 'static_pages/logged_in_home' %>
<% else %>
  <%= render 'static_pages/non_logged_in_home' %>
<% end %>
```

The test suite should now be **GREEN**:

Listing 11.7: **GREEN**

```
$ bundle exec rake test
```

It's times like this when it's worth remembering: *This is why we have tests.*

11.2 Exercise 2

The test for the microposts sidebar count first logs in as **`@user`** and checks that the string

```
"#{@user.microposts.count} microposts"
```

appears in the Home page's HTML. It then logs in as another user, one with *zero* microposts, and checks for the string

```
"0 microposts"
```

Finally, the test creates exactly *one* micropost for the other user, and then verifies that the pluralization changes by checking for the string

```
"1 micropost"
```

The result appears in Listing 11.8.

Listing 11.8: A test for the sidebar micropost count. **GREEN**
test/integration/microposts_interface_test.rb

```
require 'test_helper'

class MicropostInterfaceTest < ActionDispatch::IntegrationTest

  def setup
    @user = users(:michael)
  end
  .
  .
  .
  test "micropost sidebar count" do
    log_in_as(@user)
    get root_path
    assert_match "#{@user.microposts.count} microposts", response.body
    # User with zero microposts
    other_user = users(:mallory)
    log_in_as(other_user)
    get root_path
    assert_match "0 microposts", response.body
    # Create a micropost.
    other_user.microposts.create!(content: "A micropost")
    get root_path
    assert_match "1 micropost", response.body
  end
end
```

The test suite should now be **GREEN**:

Listing 11.9: GREEN

```
$ bundle exec rake test
```

11.3 Exercise 3

The preparation involves first configuring Git to ignore images uploaded in tests via the `.gitignore` file shown in Listing 11.10. Next, we need to create an initializer file (run automatically as part of loading Rails) to disable image resizing in tests, which causes an error:

```
$ touch config/initializers/skip_image_resizing.rb
```

Finally, we need to fill the initializer file with the contents of Listing 11.11.

Listing 11.10: Adding the uploads directory to the `.gitignore` file.

```
# See https://help.github.com/articles/ignoring-files for more about ignoring
# files.
#
# If you find yourself ignoring temporary files generated by your text editor
# or operating system, you probably want to add a global ignore instead:
#   git config --global core.excludesfile '~/.gitignore_global'

# Ignore bundler config.
/.bundle

# Ignore the default SQLite database.
/db/*.sqlite3
/db/*.sqlite3-journal

# Ignore all logfiles and tempfiles.
/log/*.log
/tmp

# Ignore Spring files.
/spring/*.pid

# Ignore uploaded test images.
/public/uploads
```

Listing 11.11: An initializer to skip image resizing in tests.
config/initializers/skip_image_resizing.rb

```
if Rails.env.test?
  CarrierWave.configure do |config|
    config.enable_processing = false
  end
end
```

Inspecting the image upload element on the Home page reveals the presence of an `input` tag with type `file`, as shown in Figure 11.1. This suggests the following assertion to check that the upload element is present on the Home page:

```
assert_select 'input[type=file]'
```

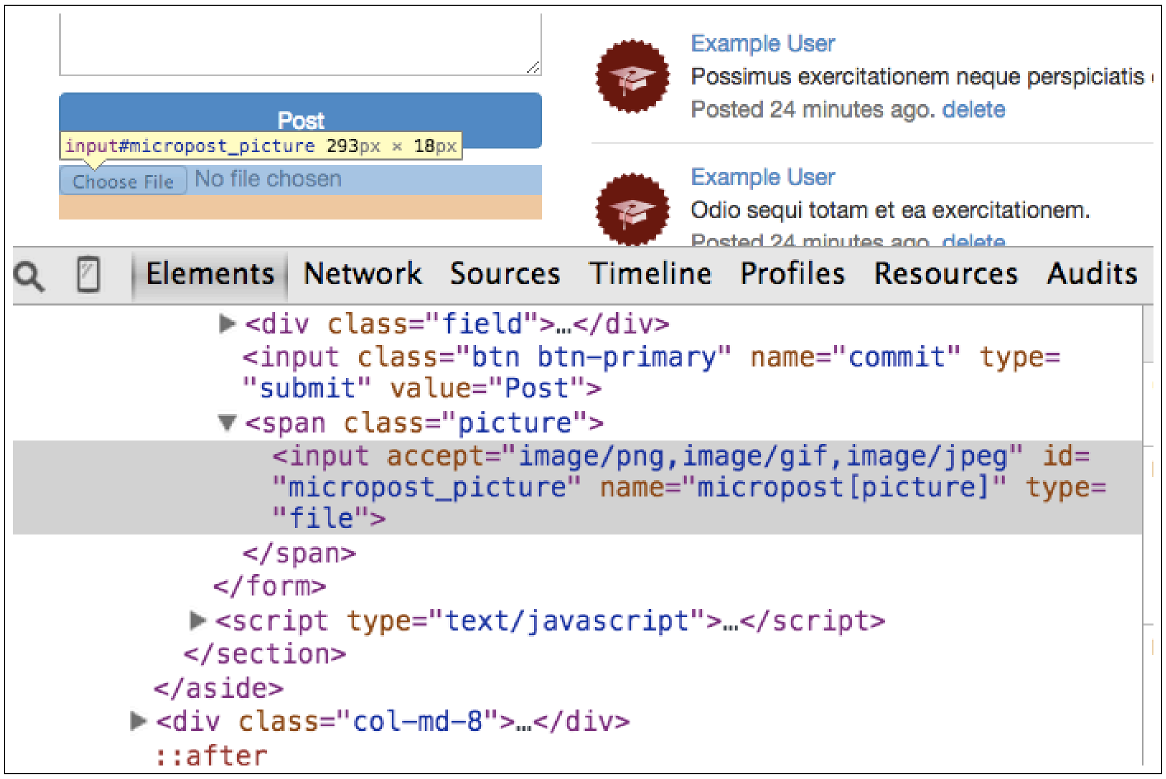
Meanwhile, we need to verify that an image is correctly uploaded. We can do this by calling `picture?` on the `@micropost` variable defined in the Microposts controller's `create` action, which we access in the test using `assigns`:

```
assert assigns(:micropost).picture?
```

Putting everything together gives the test shown in Listing 11.12.

Listing 11.12: Testing image upload. GREEN
test/integration/microposts_interface_test.rb

```
require 'test_helper'

class MicropostsInterfaceTest < ActionDispatch::IntegrationTest

  def setup
    @user = users(:michael)
  end

  test "micropost interface" do
```


Example User
Possimus exercitationem neque perspiciatis
Posted 24 minutes ago. delete
Example User
Odio sequi totam et ea exercitationem.
Posted 24 minutes ago. delete
Post
input#micropost_picture 293px × 18px
Choose File
No file chosen
Elements
Network
Sources
Timeline
Profiles
Resources
Audits

```
▶ <div class="field">…</div>
  <input class="btn btn-primary" name="commit" type=
  "submit" value="Post">
▼ <span class="picture">
    <input accept="image/png,image/gif,image/jpeg" id=
    "micropost_picture" name="micropost[picture]" type=
    "file">
  </span>
</form>
▶ <script type="text/javascript">…</script>
</section>
</aside>
▶ <div class="col-md-8">…</div>
::after
```

Figure 11.1: Inspecting the image upload element.

```
    log_in_as(@user)
    get root_path
    assert_select 'div.pagination'
    assert_select 'input[type=file]'
    # Invalid submission
    assert_no_difference 'Micropost.count' do
      post microposts_path, micropost: { content: "" }
    end
    assert_select 'div#error_explanation'
    # Valid submission
    content = "This micropost really ties the room together"
    picture = fixture_file_upload('test/fixtures/rails.png', 'image/png')
    assert_difference 'Micropost.count', 1 do
      post microposts_path, micropost: { content: content, picture: picture }
    end
    assert assigns(:micropost).picture?
    assert_redirected_to root_url
    follow_redirect!
    assert_match content, response.body
    # Delete a post.
    assert_select 'a', text: 'delete'
    first_micropost = @user.microposts.paginate(page: 1).first
    assert_difference 'Micropost.count', -1 do
      delete micropost_path(first_micropost)
    end
    # Visit a different user.
    get user_path(users(:archer))
    assert_select 'a', text: 'delete', count: 0
  end
  .
  .
  .
end
```

The test suite should now be **GREEN**:

Listing 11.13: GREEN

```
$ bundle exec rake test
```

Chapter 12

Solutions to Chapter 12 exercises

12.1 Exercise 1

To test the sidebar stats, we first verify that there are links with URLs for both the following and followers pages:

```
assert_select 'a[href=?]', following_user_path(@user)
assert_select 'a[href=?]', followers_user_path(@user)
```

Next, we test for things like `"2 following"`. The current implementation uses the `strong` tag, with different CSS ids to separate the two cases, giving HTML of the form

```
<strong id="following" class="stat">
  2
</strong>
following
```

and

```
<strong id="followers" class="stat">
  1
</strong>
followers
```

This suggests using assertions like

```
assert_select 'strong#following', @user.following.count.to_s
assert_select 'strong#followers', @user.followers.count.to_s
```

Tying the test this closely to something like the `strong` tag is brittle, though, and it would be better to use something less likely to change. Happily, the `assert_select` method still works if we omit the tag name and simply supply a CSS id, as follows:

```
assert_select '#following', @user.following.count.to_s
assert_select '#followers', @user.followers.count.to_s
```

Because the stats on the Home page and on the user's profile page both come from the same partial, there's no need to test them separately, so for convenience we'll put the stats assertions in the test for the user profile page. Applying the discussion above gives the test shown in Listing 12.1.

Listing 12.1: Adding assertions for user stats. **GREEN**
test/integration/users_profile_test.rb

```
require 'test_helper'

class UsersProfileTest < ActionDispatch::IntegrationTest
  include ApplicationHelper

  def setup
    @user = users(:michael)
  end

  test "profile display" do
    get user_path(@user)
    assert_select 'title', full_title(@user.name)
    assert_match @user.name, response.body
    assert_select 'img.gravatar'
    assert_match @user.microposts.count.to_s, response.body
    assert_select 'div.pagination'
    @user.microposts.paginate(page: 1).each do |micropost|
      assert_match micropost.content, response.body
    end
    # Following/follower stats
    assert_select 'a[href=?]', following_user_path(@user)
    assert_select 'a[href=?]', followers_user_path(@user)
    assert_select '#following', text: @user.following.count.to_s
    assert_select '#followers', text: @user.followers.count.to_s
  end
end
```

The test suite should be **GREEN**:

Listing 12.2: **GREEN**

```
$ bundle exec rake test
```

12.2 Exercise 2

Following the models provided by previous tests of the HTML page content, an initial guess at a test for the micropost content on the Home page appears in Listing 12.3.

Listing 12.3: An initial test of the feed HTML. **RED**
test/integration/following_test.rb

```
require 'test_helper'

class FollowingTest < ActionDispatch::IntegrationTest

  def setup
    @user = users(:michael)
    log_in_as(@user)
  end
  .
  .
  .
  test "feed on Home page" do
    get root_path
    @user.feed.paginate(page: 1).each do |micropost|
      assert_match micropost.content, response.body
    end
  end
end
```

The test in Listing 12.3 gets the Home page and then iterates through the first page of the feed, verifying that each micropost's content appears on the page. Unfortunately, it is somewhat mysteriously **RED**:

Listing 12.4: RED

```
$ bundle exec rake test
  .
  .
  .
  FAIL["test_feed_on_Home_page", FollowingTest, 3.025827809]
  test_feed_on_Home_page#FollowingTest (3.03s)
         Expected /I'm\ sorry\.\ Your\ words\ made\ sense,\ but\ your\
         sarcastic\ tone\ did\ not\./ to match "<!DOCTYPE html..."
  .
  .
  .
```

Here the micropost content is expected to match the HTML of the page, but apparently it doesn't. Carefully inspecting the error output shows that instead of appearing as

```
I'm sorry. Your words made sense, but your sarcastic tone did not.
```

in fact the micropost's content appears in the HTML source as follows:

```
I&#39;m sorry. Your words made sense, but your sarcastic tone did not.
```

We see that Rails has transformed the apostrophe in "I'm sorry" to the HTML escape code `&#39;`. This is because Rails *escapes* inserted HTML by default, which helps ensure that the markup is valid and thwarts malicious attacks such as cross-site scripting (XSS).

As a result of this default escaping, to get the test to pass we need to escape the micropost content before making the comparison with `assert_match`. By Googling for "ruby escape html", you can discover that the solution is to use `CGI.escapeHTML`:

```
$ rails console
>> CGI.escapeHTML("I'm sorry.")
=> "I&#39;m sorry."
```

Applying this to the test in Listing 12.3 gives the code shown in Listing 12.5.

Listing 12.5: A working test of the feed HTML. **GREEN**
test/integration/following_test.rb

```
require 'test_helper'

class FollowingTest < ActionDispatch::IntegrationTest

  def setup
    @user = users(:michael)
    log_in_as(@user)
  end
  .
  .
  .
  test "feed on Home page" do
    get root_path
    @user.feed.paginate(page: 1).each do |micropost|
      assert_match CGI.escapeHTML(micropost.content), response.body
    end
  end
end
```

As a result of the corrected test in Listing 12.5, the test suite should now be **GREEN**:

Listing 12.6: GREEN

```
$ bundle exec rake test
```