```
assert_select 'h1>img.gravatar'
```

This checks for an **img** tag with class **gravatar** *inside* a top-level heading tag (**h1**).

Because the application code was working, the test suite should be GREEN:

Listing 13.29: GREEN

```
$ rails test
```

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. Comment out the application code needed to change the two **'h1'** lines in Listing 13.28 from GREEN to RED.
2. Update Listing 13.28 to test that **will_paginate** appears only *once*. *Hint*: Refer to Table 5.2.

13.3 Manipulating microposts

Having finished both the data modeling and display templates for microposts, we now turn our attention to the interface for creating them through the web. In this section, we'll also see the first hint of a *status feed*—a notion brought to full fruition in Chapter 14. Finally, as with users, we'll make it possible to destroy microposts through the web.

There is one break with past convention worth noting: the interface to the Microposts resource will run principally through the Profile and Home pages, so we won't need actions like **new** or **edit** in the Microposts controller; we'll need only **create** and **destroy**. This leads to the routes for the Microposts

HTTP request	URL	Action	Named route
POST	/microposts	`create`	`microposts_path`
DELETE	/microposts/1	`destroy`	`micropost_path(micropost)`

Table 13.2: RESTful routes provided by the Microposts resource in Listing 13.30.

resource shown in Listing 13.30. The code in Listing 13.30 leads in turn to the RESTful routes shown in Table 13.2, which is a small subset of the full set of routes seen in Table 2.3. Of course, this simplicity is a sign of being *more* advanced, not less—we've come a long way since our reliance on scaffolding in Chapter 2, and we no longer need most of its complexity.

Listing 13.30: Routes for the Microposts resource.
config/routes.rb

```
Rails.application.routes.draw do
  root   'static_pages#home'
  get    '/help',    to: 'static_pages#help'
  get    '/about',   to: 'static_pages#about'
  get    '/contact', to: 'static_pages#contact'
  get    '/signup',  to: 'users#new'
  get    '/login',   to: 'sessions#new'
  post   '/login',   to: 'sessions#create'
  delete '/logout',  to: 'sessions#destroy'
  resources :users
  resources :account_activations, only: [:edit]
  resources :password_resets,     only: [:new, :create, :edit, :update]
  resources :microposts,          only: [:create, :destroy]
end
```

13.3.1 Micropost access control

We begin our development of the Microposts resource with some access control in the Microposts controller. In particular, because we access microposts through their associated users, both the `create` and `destroy` actions must require users to be logged in.

Tests to enforce logged-in status mirror those for the Users controller (Listing 10.20 and Listing 10.61). We simply issue the correct request to each action and confirm that the micropost count is unchanged and the result is redirected to the login URL, as seen in Listing 13.31.

Listing 13.31: Authorization tests for the Microposts controller. RED
test/controllers/microposts_controller_test.rb

```
require 'test_helper'

class MicropostsControllerTest < ActionDispatch::IntegrationTest

  def setup
    @micropost = microposts(:orange)
  end

  test "should redirect create when not logged in" do
    assert_no_difference 'Micropost.count' do
      post microposts_path, params: { micropost: { content: "Lorem ipsum" } }
    end
    assert_redirected_to login_url
  end

  test "should redirect destroy when not logged in" do
    assert_no_difference 'Micropost.count' do
      delete micropost_path(@micropost)
    end
    assert_redirected_to login_url
  end
end
```

Writing the application code needed to get the tests in Listing 13.31 to pass requires a little refactoring first. Recall from Section 10.2.1 that we enforced the login requirement using a before filter that called the **`logged_in_user`** method (Listing 10.15). At the time, we needed that method only in the Users controller, but now we find that we need it in the Microposts controller as well, so we'll move it into the Application controller, which is the base class of all controllers (Section 4.4.4).[11] The result appears in Listing 13.32.

[11] Note that, unlike the behavior in languages like Java or C++, private methods in Ruby can be called from derived classes. Thanks to reader Vishal Antony for bringing this difference to my attention.

Listing 13.32: Moving the `logged_in_user` method into the Application controller. RED

app/controllers/application_controller.rb

```
class ApplicationController < ActionController::Base
  include SessionsHelper

  private

    # Confirms a logged-in user.
    def logged_in_user
      unless logged_in?
        store_location
        flash[:danger] = "Please log in."
        redirect_to login_url
      end
    end
end
```

To avoid code repetition, you should also remove `logged_in_user` from the Users controller at this time (Listing 13.33).

Listing 13.33: The Users controller with the logged-in user filter removed. RED

app/controllers/users_controller.rb

```
class UsersController < ApplicationController
  before_action :logged_in_user, only: [:index, :edit, :update, :destroy]
  .
  .
  .
  private

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

    # Before filters

    # Confirms the correct user.
    def correct_user
      @user = User.find(params[:id])
      redirect_to(root_url) unless current_user?(@user)
    end
```

```
    # Confirms an admin user.
    def admin_user
      redirect_to(root_url) unless current_user.admin?
    end
end
```

With the code in Listing 13.32, the **`logged_in_user`** method is now available in the Microposts controller, which means that we can add **`create`** and **`destroy`** actions and then restrict access to them using a before filter, as shown in Listing 13.34.

Listing 13.34: Adding authorization to the Microposts controller actions. GREEN

app/controllers/microposts_controller.rb

```
class MicropostsController < ApplicationController
  before_action :logged_in_user, only: [:create, :destroy]

  def create
  end

  def destroy
  end
end
```

At this point, the tests should pass:

Listing 13.35: GREEN

```
$ rails test
```

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. Why is it a bad idea to leave a copy of **`logged_in_user`** in the Users controller?

13.3.2 Creating microposts

In Chapter 7, we implemented user signup by making an HTML form that issued an HTTP `POST` request to the **`create`** action in the Users controller. The implementation of micropost creation is similar; the main difference is that, rather than using a separate page at /microposts/new, we will put the form on the Home page itself (i.e., the root path /), as mocked up in Figure 13.10.

When we last left the Home page, it appeared as in Figure 5.8—that is, it had a "Sign up now!" button in the middle. Since a micropost creation form makes sense only in the context of a particular logged-in user, one goal of this section will be to serve different versions of the Home page depending on a visitor's login status. We'll implement this in Listing 13.37 below.

We'll start with the **`create`** action for microposts, which is similar to its user analogue (Listing 7.26); the principal difference lies in using the user/micropost association to **`build`** the new micropost, as seen in Listing 13.36. Note the use of strong parameters via **`micropost_params`**, which permits only the micropost's **`content`** attribute to be modified through the web.

Listing 13.36: The Microposts controller **`create`** action.
app/controllers/microposts_controller.rb

```
class MicropostsController < ApplicationController
  before_action :logged_in_user, only: [:create, :destroy]

  def create
    @micropost = current_user.microposts.build(micropost_params)
    if @micropost.save
      flash[:success] = "Micropost created!"
      redirect_to root_url
    else
      render 'static_pages/home'
    end
  end

  def destroy
  end

  private

    def micropost_params
      params.require(:micropost).permit(:content)
    end
end
```

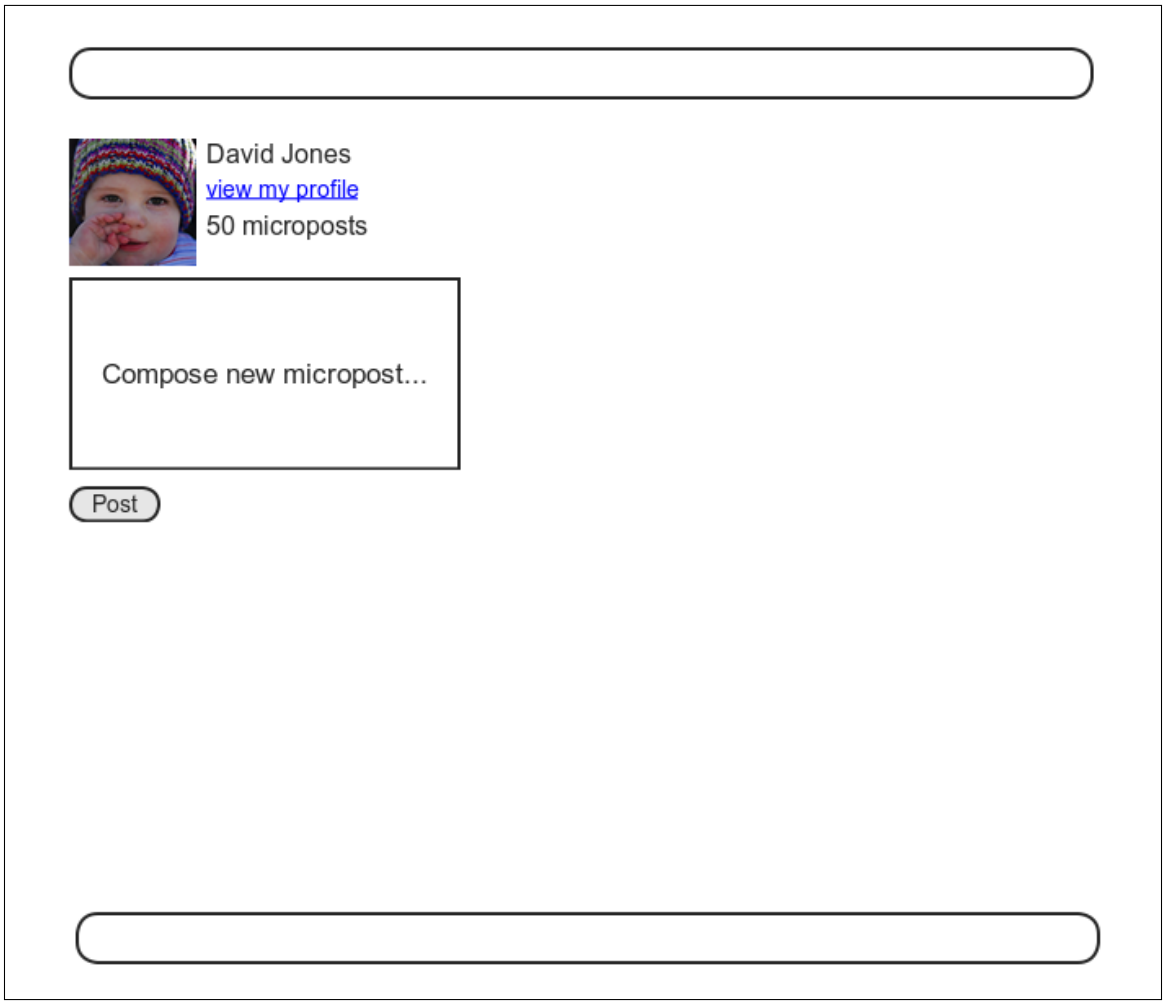

Figure 13.10: A mockup of the Home page with a form for creating microposts.

To build a form for creating microposts, we use the code in Listing 13.37, which serves up different HTML based on whether the site visitor is logged in or not.

Listing 13.37: Adding microposts creation to 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>
<% else %>
  <div class="center jumbotron">
    <h1>Welcome to the Sample App</h1>

    <h2>
      This is the home page for the
      <a href="https://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.svg", alt: "Rails logo", width: "200"),
                        "https://rubyonrails.org/" %>
<% end %>
```

(Having so much code in each branch of the **`if-else`** conditional is a bit messy, and cleaning it up using partials is left as an exercise (Section 13.3.2).)

To get the page defined in Listing 13.37 working, we need to create and fill in a couple of partials. The first is the new Home page sidebar, as shown in Listing 13.38.

Listing 13.38: The partial for the user info sidebar.
app/views/shared/_user_info.html.erb

```
<%= link_to gravatar_for(current_user, size: 50), current_user %>
<h1><%= current_user.name %></h1>
<span><%= link_to "view my profile", current_user %></span>
<span><%= pluralize(current_user.microposts.count, "micropost") %></span>
```

Note that, as in the profile sidebar (Listing 13.24), the user info in Listing 13.38 displays the total number of microposts for the user. There's a slight difference in the display, though; in the profile sidebar, "Microposts" is a label, and showing "Microposts (1)" makes sense. In the present case, though, saying "1 microposts" is ungrammatical, so we arrange to display "1 micropost" and "2 microposts" using the **`pluralize`** method we saw in Section 7.3.3.

We next define the form for creating microposts (Listing 13.39), which is similar to the signup form in Listing 7.15.

Listing 13.39: The form partial for creating microposts.
app/views/shared/_micropost_form.html.erb

```
<%= form_with(model: @micropost, local: true) do |f| %>
  <%= render 'shared/error_messages', object: f.object %>
  <div class="field">
    <%= f.text_area :content, placeholder: "Compose new micropost..." %>
  </div>
  <%= f.submit "Post", class: "btn btn-primary" %>
<% end %>
```

We need to make two changes before the form in Listing 13.39 will work. First, we need to define **`@micropost`**, which (as before) we do through the association:

```
@micropost = current_user.microposts.build
```

The result appears in Listing 13.40.

Listing 13.40: Adding a micropost instance variable to the **`home`** action.
app/controllers/static_pages_controller.rb

```
class StaticPagesController < ApplicationController

  def home
    @micropost = current_user.microposts.build if logged_in?
  end

  def help
  end

  def about
  end

  def contact
  end
end
```

Of course, **current_user** exists only if the user is logged in, so the **@micropost** variable should only be defined in this case.

The second change needed to get Listing 13.39 to work is to redefine the error-messages partial so the following code from Listing 13.39 works:

```
<%= render 'shared/error_messages', object: f.object %>
```

You may recall from Listing 7.20 that the error-messages partial references the **@user** variable explicitly, but in the present case we have an **@micropost** variable instead. To unify these cases, we can pass the form variable **f** to the partial and access the associated object through **f.object**, so that in

```
form_with(model: @user, local: true) do |f|
```

f.object is **@user**, and in

```
form_with(model: @micropost, local: true) do |f|
```

f.object is **@micropost**, etc.

To pass the object to the partial, we use a hash with value equal to the object and key equal to the desired name of the variable in the partial, which is

what the second line in Listing 13.39 accomplishes. In other words, **`object: f.object`** creates a variable called **`object`** in the **`error_messages`** partial, and we can use it to construct a customized error message, as shown in Listing 13.41.

Listing 13.41: Error messages that work with other objects. RED
app/views/shared/_error_messages.html.erb

```
<% if object.errors.any? %>
  <div id="error_explanation">
    <div class="alert alert-danger">
      The form contains <%= pluralize(object.errors.count, "error") %>.
    </div>
    <ul>
    <% object.errors.full_messages.each do |msg| %>
      <li><%= msg %></li>
    <% end %>
    </ul>
  </div>
<% end %>
```

At this point, you should verify that the test suite is RED:

Listing 13.42: RED

```
$ rails test
```

This is a hint that we need to update the other occurrences of the error-messages partial, which we used when signing up users (Listing 7.20), resetting passwords (Listing 12.14), and editing users (Listing 10.2). The updated versions are shown in Listing 13.43, Listing 13.45, and Listing 13.44.

Listing 13.43: Updating the rendering of user signup errors. RED
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_with(model: @user, local: true) do |f| %>
      <%= render 'shared/error_messages', object: f.object %>

      <%= 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' %>

      <%= f.submit "Create my account", class: "btn btn-primary" %>
    <% end %>
  </div>
</div>
```

Listing 13.44: Updating the errors for editing users. RED
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_with(model: @user, local: true) do |f| %>
      <%= render 'shared/error_messages', object: f.object %>

      <%= 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' %>

      <%= f.submit "Save changes", class: "btn btn-primary" %>
    <% end %>

    <div class="gravatar_edit">
      <%= gravatar_for @user %>
```

```
      <a href="https://gravatar.com/emails">change</a>
    </div>
  </div>
</div>
```

Listing 13.45: Updating the errors for password resets. GREEN
app/views/password_resets/edit.html.erb

```
<% provide(:title, 'Reset password') %>
<h1>Reset password</h1>

<div class="row">
  <div class="col-md-6 col-md-offset-3">
    <%= form_with(model: @user, url: password_reset_path(params[:id]),
                  local: true) do |f| %>
      <%= render 'shared/error_messages', object: f.object %>

      <%= hidden_field_tag :email, @user.email %>

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

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

      <%= f.submit "Update password", class: "btn btn-primary" %>
    <% end %>
  </div>
</div>
```

At this point, all the tests should be GREEN:

```
$ rails test
```

Additionally, all the HTML in this section should render properly, showing the form as in Figure 13.11, and a form with a submission error as in Figure 13.12.

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.

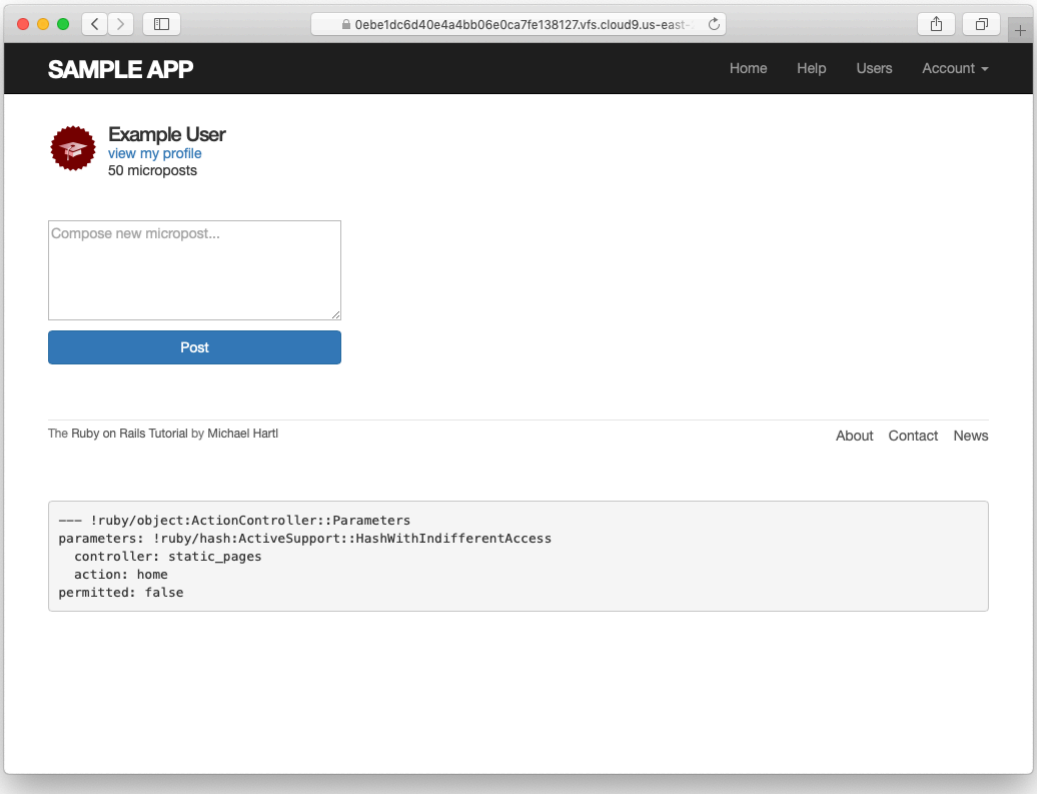

Figure 13.11: The Home page with a new micropost form.

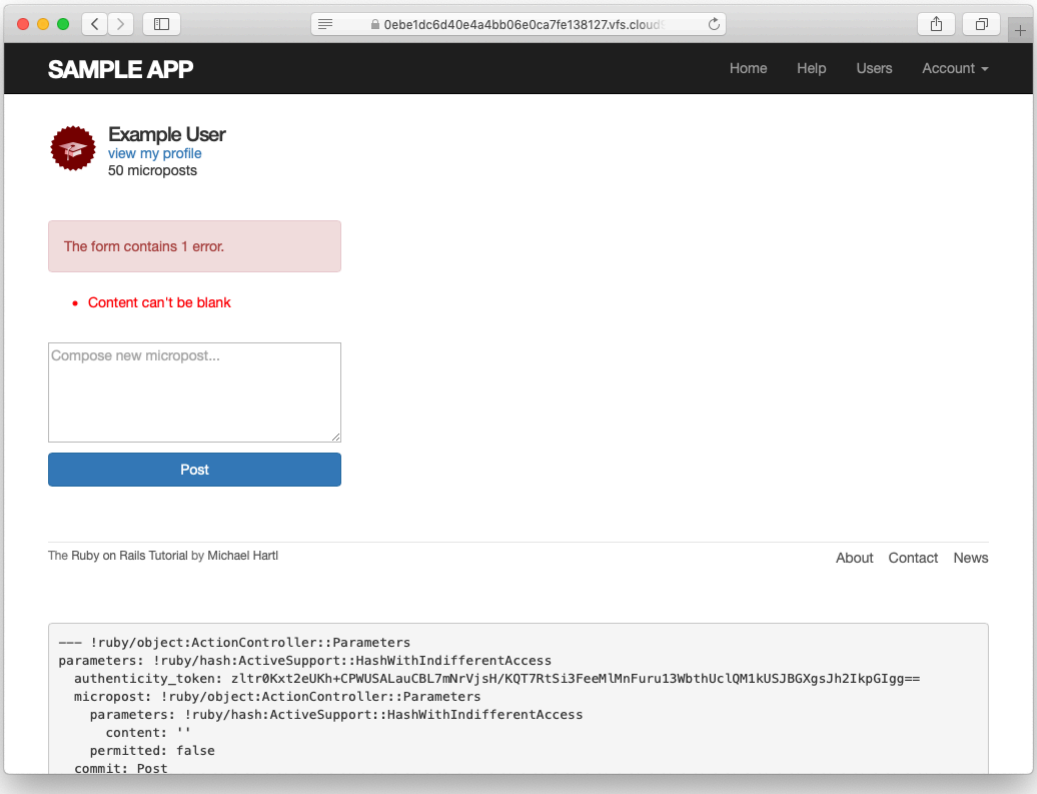

Figure 13.12: The Home page with a form error.

1. Refactor the Home page to use separate partials for the two branches of the `if-else` statement.

13.3.3 A proto-feed

Although the micropost form is actually now working, users can't immediately see the results of a successful submission because the current Home page doesn't display any microposts. If you like, you can verify that the form shown in Figure 13.11 is working by submitting a valid entry and then navigating to the profile page to see the post, but that's rather cumbersome. It would be far better to have a *feed* of microposts that includes the user's own posts, as mocked up in Figure 13.13. (In Chapter 14, we'll generalize this feed to include the microposts of users being *followed* by the current user, à la Twitter.)

Since each user should have a feed, we are led naturally to a `feed` method in the User model, which will initially just select all the microposts belonging to the current user. We'll accomplish this using the `where` method on the `Micropost` model (seen briefly before in Section 11.3.3), as shown in Listing 13.46.[12]

Listing 13.46: A preliminary implementation for the micropost status feed.
app/models/user.rb

```
class User < ApplicationRecord
  .
  .
  .
  # Defines a proto-feed.
  # See "Following users" for the full implementation.
  def feed
    Micropost.where("user_id = ?", id)
  end

  private
    .
    .
    .
end
```

The question mark in

[12] See the Rails Guide on the Active Record Query Interface for more on `where` and related methods.

Figure 13.13: A mockup of the Home page with a proto-feed.

```
Micropost.where("user_id = ?", id)
```

ensures that **id** is properly *escaped* before being included in the underlying SQL query, thereby avoiding a serious security hole called *SQL injection*. The **id** attribute here is just an integer (i.e., **self.id**, the unique ID of the user), so there is no danger of SQL injection in this case, but it does no harm, and *always* escaping variables injected into SQL statements is a good habit to cultivate.

Alert readers might note at this point that the code in Listing 13.46 is essentially equivalent to writing

```
def feed
  microposts
end
```

We've used the code in Listing 13.46 instead because it generalizes much more naturally to the full status feed needed in Chapter 14.

To use the feed in the sample application, we add an **@feed_items** instance variable for the current user's (paginated) feed, as in Listing 13.47, and then add a status feed partial (Listing 13.48) to the Home page (Listing 13.49). Note that, now that there are two lines that need to be run when the user is logged in, Listing 13.47 changes

```
@micropost = current_user.microposts.build if logged_in?
```

from Listing 13.40 to

```
  if logged_in?
    @micropost  = current_user.microposts.build
    @feed_items = current_user.feed.paginate(page: params[:page])
  end
```

thereby moving the conditional from the end of the line to an if-end statement.

Listing 13.47: Adding a feed instance variable to the **home** action.
app/controllers/static_pages_controller.rb

```
class StaticPagesController < ApplicationController

  def home
    if logged_in?
      @micropost  = current_user.microposts.build
      @feed_items = current_user.feed.paginate(page: params[:page])
    end
  end

  def help
  end

  def about
  end

  def contact
  end
end
```

Listing 13.48: The status feed partial.
app/views/shared/_feed.html.erb

```
<% if @feed_items.any? %>
  <ol class="microposts">
    <%= render @feed_items %>
  </ol>
  <%= will_paginate @feed_items %>
<% end %>
```

The status feed partial defers the rendering to the micropost partial defined in Listing 13.22:

```
<%= render @feed_items %>
```

Here Rails knows to call the micropost partial because each element of **@feed_items** has class **Micropost**. This causes Rails to look for a partial with the corresponding name in the views directory of the given resource:

app/views/microposts/_micropost.html.erb

We can add the feed to the Home page by rendering the feed partial as usual (Listing 13.49). The result is a display of the feed on the Home page, as required (Figure 13.14).

Listing 13.49: Adding a status feed to 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 %>
  .
  .
  .
<% end %>
```

At this point, creating a new micropost works as expected, as seen in Figure 13.15.

There is one subtlety, though: on *failed* micropost submission, the Home page expects an **`@feed_items`** instance variable, so failed submissions currently break. The solution is to create the necessary feed variable in the branch for failed submissions in the Microposts controller **`create`** action, as shown in Listing 13.50.

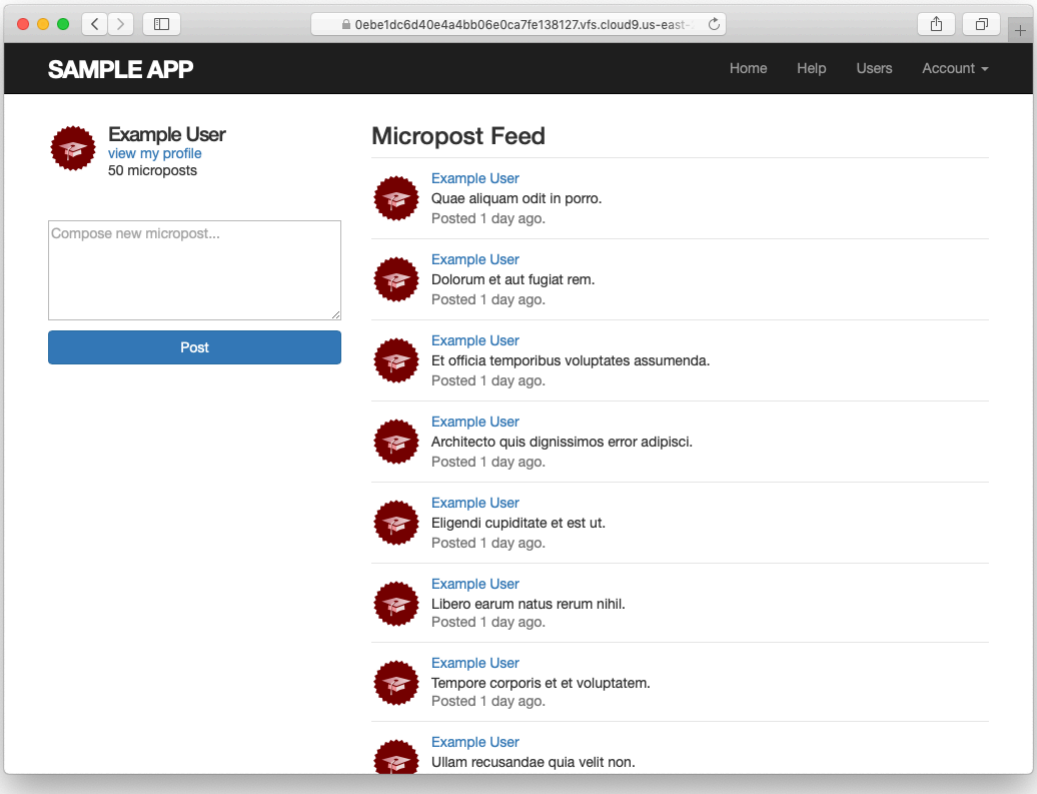

Figure 13.14: The Home page with a proto-feed.

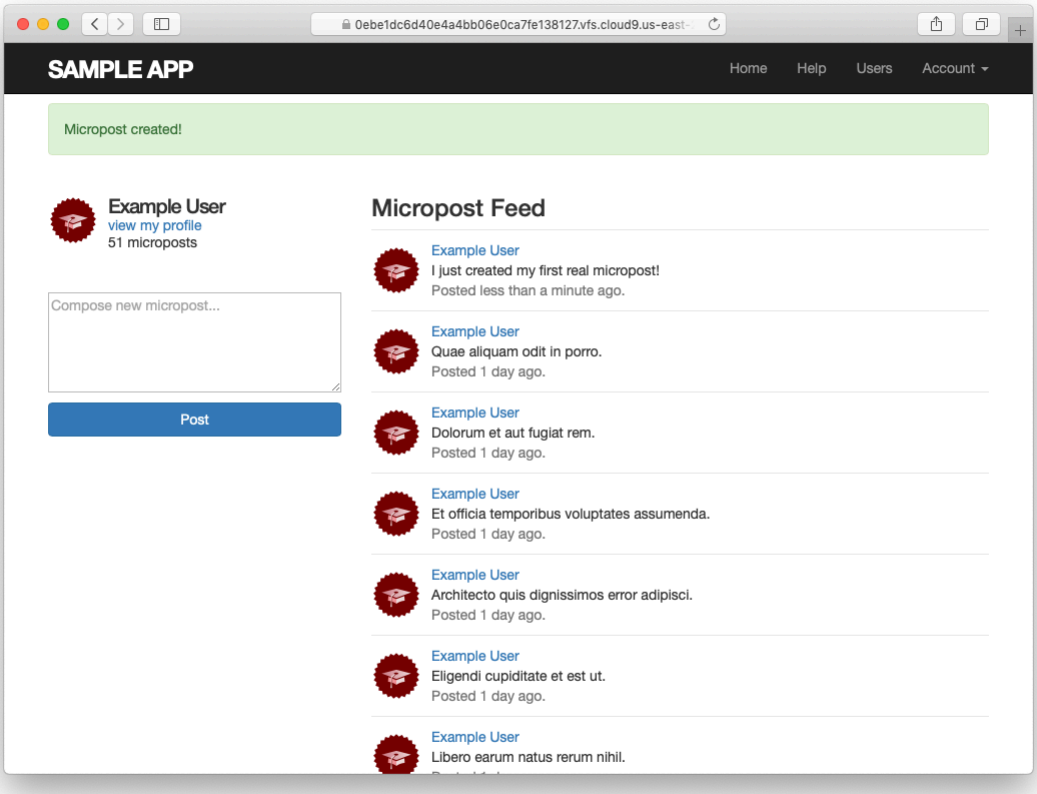

Figure 13.15: The Home page after creating a new micropost.

Listing 13.50: Adding an (empty) `@feed_items` instance variable to the `create` action.
app/controllers/microposts_controller.rb

```
class MicropostsController < ApplicationController
  before_action :logged_in_user, only: [:create, :destroy]

  def create
    @micropost = current_user.microposts.build(micropost_params)
    if @micropost.save
      flash[:success] = "Micropost created!"
      redirect_to root_url
    else
      @feed_items = current_user.feed.paginate(page: params[:page])
      render 'static_pages/home'
    end
  end

  def destroy
  end

  private

    def micropost_params
      params.require(:micropost).permit(:content)
    end
end
```

Unfortunately, pagination still doesn't quite work. We can see why by submitting an invalid micropost, say, one whose length is too long (Figure 13.16).

Scrolling down to the pagination links, we see links on both "2" and "Next" pointing to the next page (Figure 13.17). Because the `create` action is in the Microposts controller (Listing 13.50), the URL is /microposts?page=2, which tries to go to the nonexistent Microposts index action. As a result, clicking on either link gives a routing error (Figure 13.18).

We can solve this problem by giving `will_paginate` explicit `controller` and `action` parameters corresponding to the Home page, i.e., the `static_pages` controller and the `home` action.[13] The result appears in Listing 13.51.

[13] Thanks to reader Martin Francl for pointing out this solution.

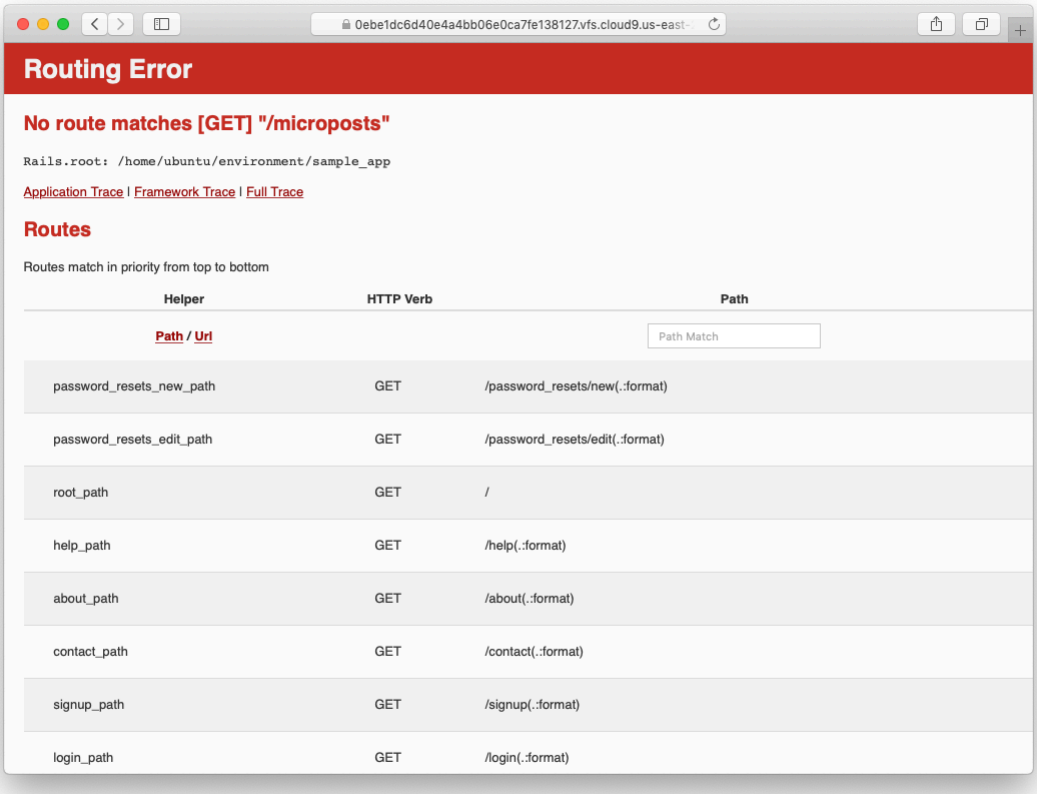

Figure 13.16: An invalid micropost on the Home page.

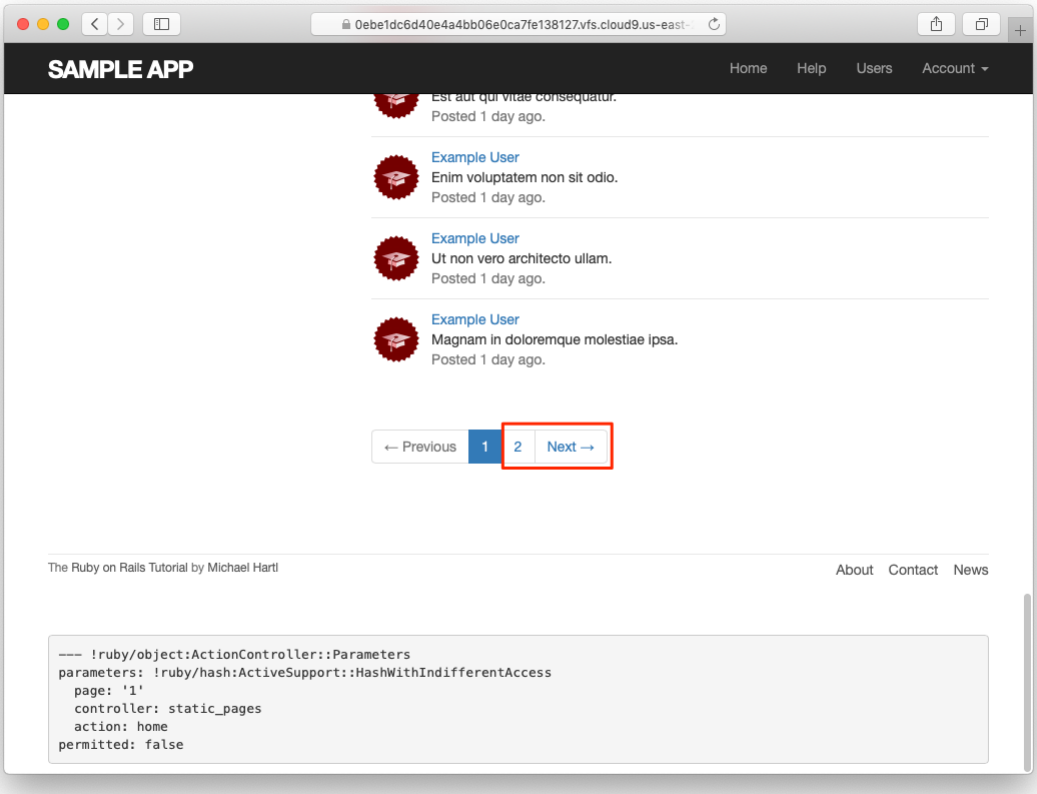

Figure 13.17: The next link on the Home page.

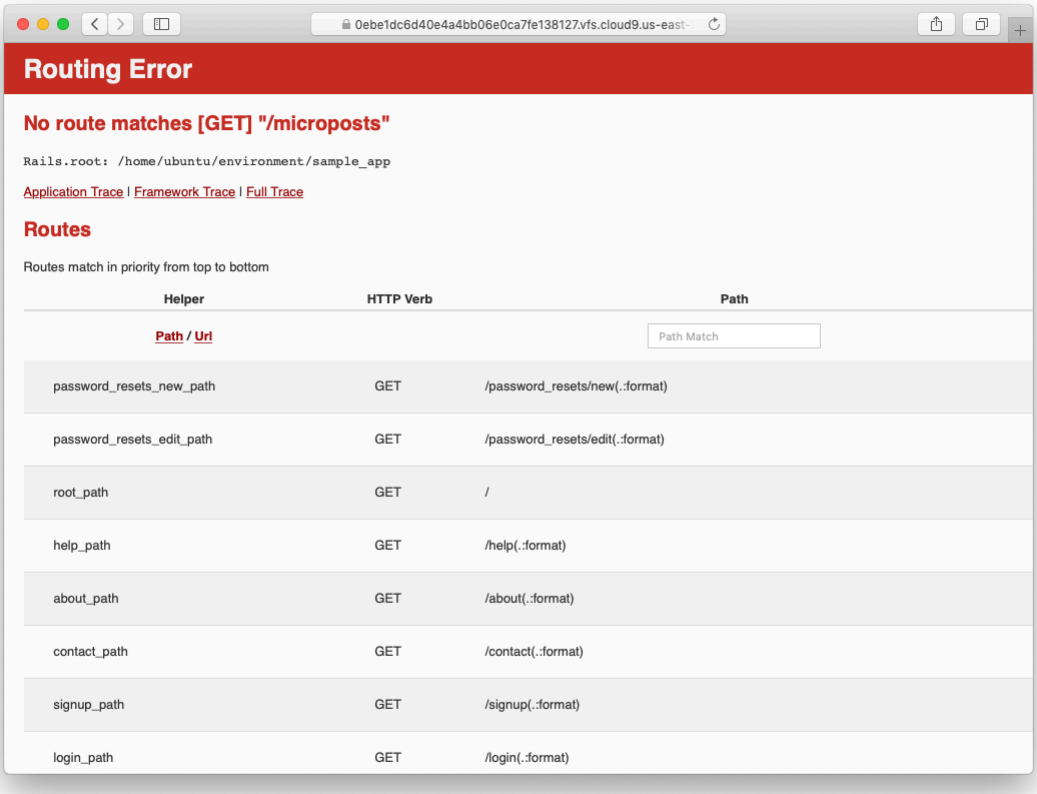

Figure 13.18: A routing error on page 2.

Listing 13.51: Setting an explicit controller and action.
app/views/shared/_feed.html.erb

```
<% if @feed_items.any? %>
  <ol class="microposts">
    <%= render @feed_items %>
  </ol>
  <%= will_paginate @feed_items,
                    params: { controller: :static_pages, action: :home } %>
<% end %>
```

Now clicking on either of the pagination links in Figure 13.17 yields the expected second page, as shown in Figure 13.19.

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. Use the newly created micropost UI to create the first real micropost. What are the contents of the `INSERT` command in the server log?
2. In the console, set `user` to the first user in the database. Confirm that the values of `Micropost.where("user_id = ?", user.id)`, `user.microposts`, and `user.feed` are all the same. *Hint*: It's probably easiest to compare directly using `==`.

13.3.4 Destroying microposts

The last piece of functionality to add to the Microposts resource is the ability to destroy posts. As with user deletion (Section 10.4.2), we accomplish this with "delete" links, as mocked up in Figure 13.20. Unlike that case, which restricted user destruction to admin users, the delete links will work only for microposts created by the current user.

Our first step is to add a delete link to the micropost partial as in Listing 13.22. The result appears in Listing 13.52.