13.2 Showing microposts

Although we don't yet have a way to create microposts through the web—that comes in Section 13.3.2—this won't stop us from displaying them (and testing that display). Following Twitter's lead, we'll plan to display a user's microposts not on a separate microposts **`index`** page but rather directly on the user **`show`** page itself, as mocked up in Figure 13.4. We'll start with fairly simple ERb templates for adding a micropost display to the user profile, and then we'll add microposts to the seed data from Section 10.3.2 so that we have something to display.

13.2.1 Rendering microposts

Our plan is to display the microposts for each user on their respective profile page (**`show.html.erb`**), together with a running count of how many microposts they've made. As we'll see, many of the ideas are similar to our work in Section 10.3 on showing all users.

In case you've added some microposts in the exercises, it's a good idea to reset and reseed the database at this time:

```
$ rails db:migrate:reset
$ rails db:seed
```

Although we won't need the Microposts controller until Section 13.3, we will need the views directory in just a moment, so let's generate the controller now:

```
$ rails generate controller Microposts
```

Our primary purpose in this section is to render all the microposts for each user. We saw in Section 10.3.5 that the code

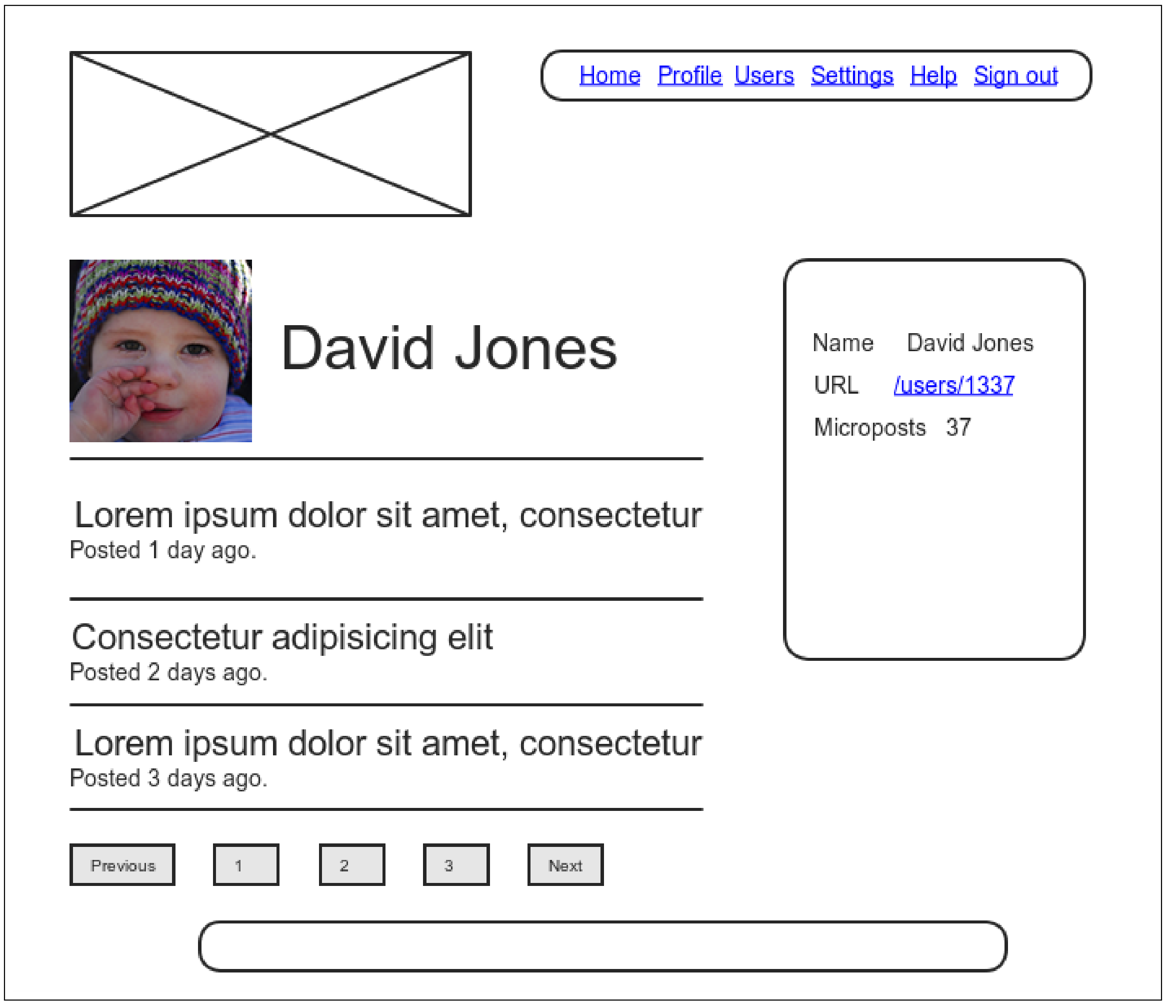


Figure 13.4: A mockup of a profile page with microposts.

```
<ul class="users">
  <%= render @users %>
</ul>
```

automatically renders each of the users in the **`@users`** variable using the **`_user.html.erb`** partial. We'll define an analogous **`_micropost.html.erb`** partial so that we can use the same technique on a collection of microposts as follows:

```
<ol class="microposts">
  <%= render @microposts %>
</ol>
```

Note that we've used the *ordered list* tag **`ol`** (as opposed to an unordered list **`ul`**) because microposts are listed in a particular order (reverse-chronological). The corresponding partial appears in Listing 13.22.

Listing 13.22: A partial for showing a single micropost.
`app/views/microposts/_micropost.html.erb`

```
<li id="micropost-<%= micropost.id %>">
  <%= link_to gravatar_for(micropost.user, size: 50), micropost.user %>
  <span class="user"><%= link_to micropost.user.name, micropost.user %></span>
  <span class="content"><%= micropost.content %></span>
  <span class="timestamp">
    Posted <%= time_ago_in_words(micropost.created_at) %> ago.
  </span>
</li>
```

This uses the awesome **`time_ago_in_words`** helper method, whose meaning is probably clear and whose effect we will see in Section 13.2.2. Listing 13.22 also adds a CSS id for each micropost using

```
<li id="micropost-<%= micropost.id %>">
```

This is a generally good practice, as it opens up the possibility of manipulating individual microposts at a future date (using JavaScript, for example).

The next step is to address the difficulty of displaying a potentially large number of microposts. We'll solve this problem the same way we solved it for users in Section 10.3.3, namely, using pagination. As before, we'll use the **`will_paginate`** method:

```
<%= will_paginate @microposts %>
```

If you compare this with the analogous line on the user index page, Listing 10.45, you'll see that before we had just

```
<%= will_paginate %>
```

This worked because, in the context of the Users controller, **`will_paginate`** *assumes* the existence of an instance variable called **`@users`** (which, as we saw in Section 10.3.3, should be of class **`ActiveRecord::Relation`**). In the present case, since we are still in the Users controller but want to paginate *microposts* instead, we'll pass an explicit **`@microposts`** variable to **`will_paginate`**. Of course, this means that we will have to define such a variable in the user **`show`** action (Listing 13.23).

Listing 13.23: Adding an **`@microposts`** instance variable to the user **`show`** action.
`app/controllers/users_controller.rb`

```
class UsersController < ApplicationController
  .
  .
  .
  def show
    @user = User.find(params[:id])
    @microposts = @user.microposts.paginate(page: params[:page])
  end
  .
  .
  .
end
```

Notice here how clever **`paginate`** is—it even works *through* the microposts association, reaching into the `microposts` table and pulling out the desired page of microposts.

Our final task is to display the number of microposts for each user, which we can do with the **`count`** method:

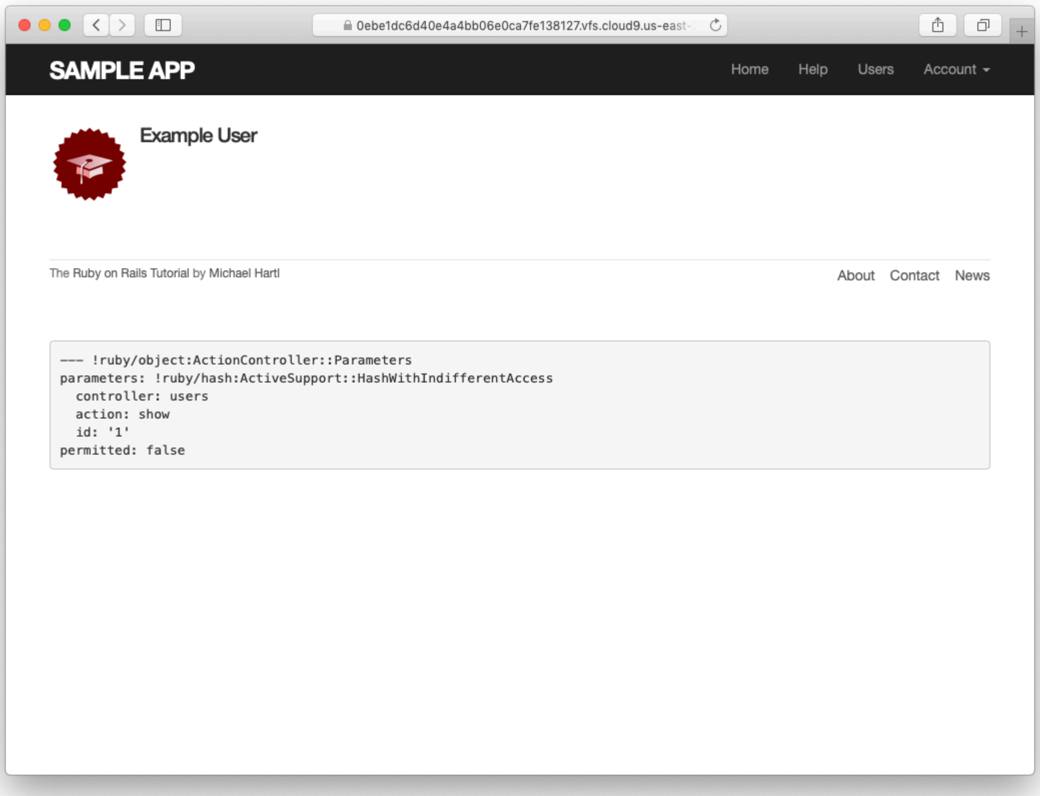
```
user.microposts.count
```

As with **`paginate`**, we can use the **`count`** method through the association. In particular, **`count`** does *not* pull all the microposts out of the database and then call **`length`** on the resulting array, as this would become inefficient as the number of microposts grew. Instead, it performs the calculation directly in the database, asking the database to count the microposts with the given **`user_id`** (an operation for which all databases are highly optimized). (In the unlikely event that finding the count is still a bottleneck in your application, you can make it even faster using a *counter cache*.)

Putting all the elements above together, we are now in a position to add microposts to the profile page, as shown in Listing 13.24. Note the use of **`if @user.microposts.any?`** (a construction we saw before in Listing 7.21), which makes sure that an empty list won't be displayed when the user has no microposts.

Listing 13.24: Adding microposts to the user **`show`** page.
`app/views/users/show.html.erb`

```
<% provide(:title, @user.name) %>
<div class="row">
  <aside class="col-md-4">
    <section class="user_info">
      <h1>
        <%= gravatar_for @user %>
        <%= @user.name %>
      </h1>
    </section>
  </aside>
  <div class="col-md-8">
    <% if @user.microposts.any? %>
      <h3>Microposts (<%= @user.microposts.count %>)</h3>
```


0ebe1dc6d40e4a4bb06e0ca7fe138127.vfs.cloud9.us-east-
SAMPLE APP
Home
Help
Users
Account
Example User
The Ruby on Rails Tutorial by Michael Hartl
About
Contact
News

```
--- !ruby/object:ActionController::Parameters
parameters: !ruby/hash:ActiveSupport::HashWithIndifferentAccess
  controller: users
  action: show
  id: '1'
permitted: false
```

Figure 13.5: The user profile page with code for microposts—but no microposts.

```
      <ol class="microposts">
        <%= render @microposts %>
      </ol>
      <%= will_paginate @microposts %>
    <% end %>
  </div>
</div>
```

At this point, we can get a look at our updated user profile page in Figure 13.5. It's rather…disappointing. Of course, this is because there are not currently any microposts. It's time to change that.

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. As mentioned briefly in Section 7.3.3, helper methods like **`time_ago_in_words`** are available in the Rails console via the **`helper`** object. Using **`helper`**, apply **`time_ago_in_words`** to **`3.weeks.ago`** and **`6.months.ago`**.

2. What is the result of **`helper.time_ago_in_words(1.year.ago)`**?

3. What is the Ruby class for a page of microposts? *Hint*: Use the code in Listing 13.23 as your model, and call the **`class`** method on **`paginate`** with the argument **`page: nil`**.

13.2.2 Sample microposts

With all the work making templates for user microposts in Section 13.2.1, the ending was rather anticlimactic. We can rectify this sad situation by adding microposts to the seed data from Section 10.3.2.

Adding sample microposts for *all* the users actually takes a rather long time, so first we'll select just the first six users (i.e., the five users with custom Gravatars, and one with the default Gravatar) using the **`take`** method:

```
User.order(:created_at).take(6)
```

The call to **`order`** ensures that we find the first six users that were created.

For each of the selected users, we'll make 50 microposts (plenty to overflow the pagination limit of 30). To generate sample content for each micropost, we'll use the Faker gem's handy `Lorem.sentence` method.[7] The result is

[7] **`Faker::Lorem.sentence`** returns *lorem ipsum* text; as noted in Chapter 6, *lorem ipsum* has a fascinating back story.

the new seed data method shown in Listing 13.25. (The reason for the order of the loops in Listing 13.25 is to intermix the microposts for use in the status feed (Section 14.3). Looping over the users first gives feeds with big runs of microposts from the same user, which is visually unappealing.)

Listing 13.25: Adding microposts to the sample data.
db/seeds.rb

```
.
.
.
# Generate microposts for a subset of users.
users = User.order(:created_at).take(6)
50.times do
  content = Faker::Lorem.sentence(word_count: 5)
  users.each { |user| user.microposts.create!(content: content) }
end
```

At this point, we can reseed the development database as usual:

```
$ rails db:migrate:reset
$ rails db:seed
```

You should also quit and restart the Rails development server.

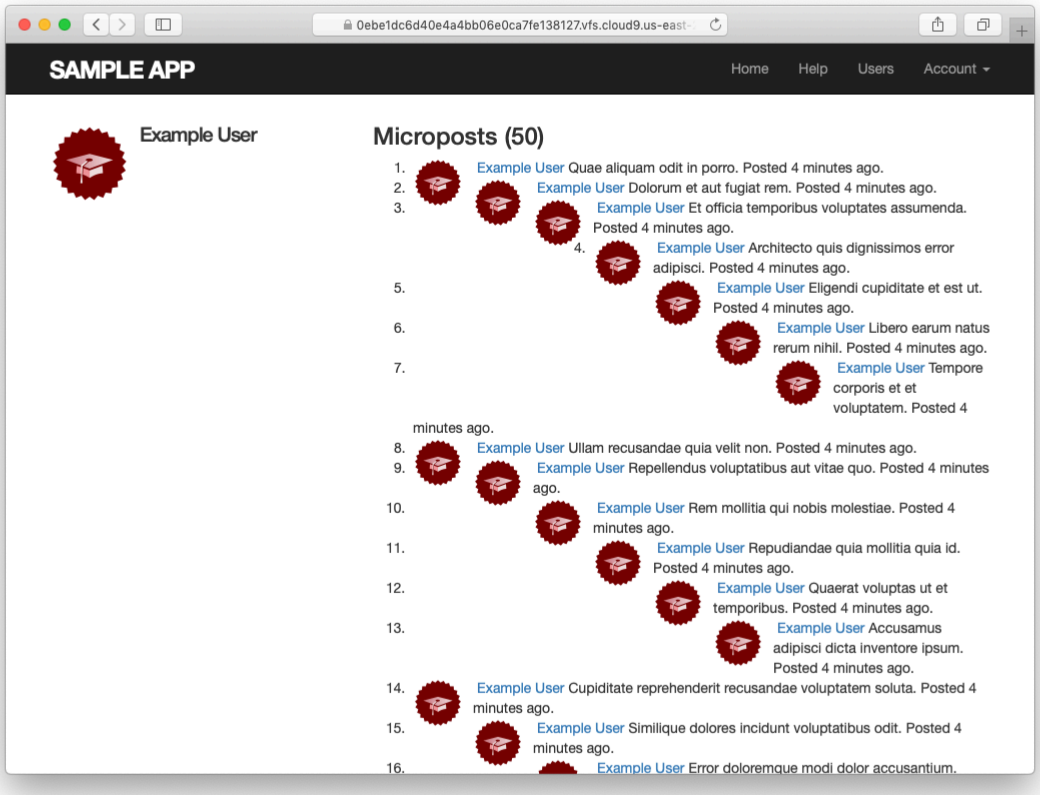
With that, we are in a position to enjoy the fruits of our Section 13.2.1 labors by displaying information for each micropost.[8] The preliminary results appear in Figure 13.6.

The page shown in Figure 13.6 has no micropost-specific styling, so let's add some (Listing 13.26) and take a look at the resulting pages.[9]

Listing 13.26: The CSS for microposts (including all the CSS for this chapter).
app/assets/stylesheets/custom.scss

[8]By design, the Faker gem's *lorem ipsum* text is randomized, so the contents of your sample microposts will differ.

[9]For convenience, Listing 13.26 actually has *all* the CSS needed for this chapter.



Example User

Microposts (50)

1. Example User Quae aliquam odit in porro. Posted 4 minutes ago.
2. Example User Dolorum et aut fugiat rem. Posted 4 minutes ago.
3. Example User Et officia temporibus voluptates assumenda. Posted 4 minutes ago.
4. Example User Architecto quis dignissimos error adipisci. Posted 4 minutes ago.
5. Example User Eligendi cupiditate et est ut. Posted 4 minutes ago.
6. Example User Libero earum natus rerum nihil. Posted 4 minutes ago.
7. Example User Tempore corporis et et voluptatem. Posted 4 minutes ago.
8. Example User Ullam recusandae quia velit non. Posted 4 minutes ago.
9. Example User Repellendus voluptatibus aut vitae quo. Posted 4 minutes ago.
10. Example User Rem mollitia qui nobis molestiae. Posted 4 minutes ago.
11. Example User Repudiandae quia mollitia quia id. Posted 4 minutes ago.
12. Example User Quaerat voluptas ut et temporibus. Posted 4 minutes ago.
13. Example User Accusamus adipisci dicta inventore ipsum. Posted 4 minutes ago.
14. Example User Cupiditate reprehenderit recusandae voluptatem soluta. Posted 4 minutes ago.
15. Example User Similique dolores incidunt voluptatibus odit. Posted 4 minutes ago.
16. Example User Error doloremque modi dolor accusantium.

Figure 13.6: The user profile with unstyled microposts.

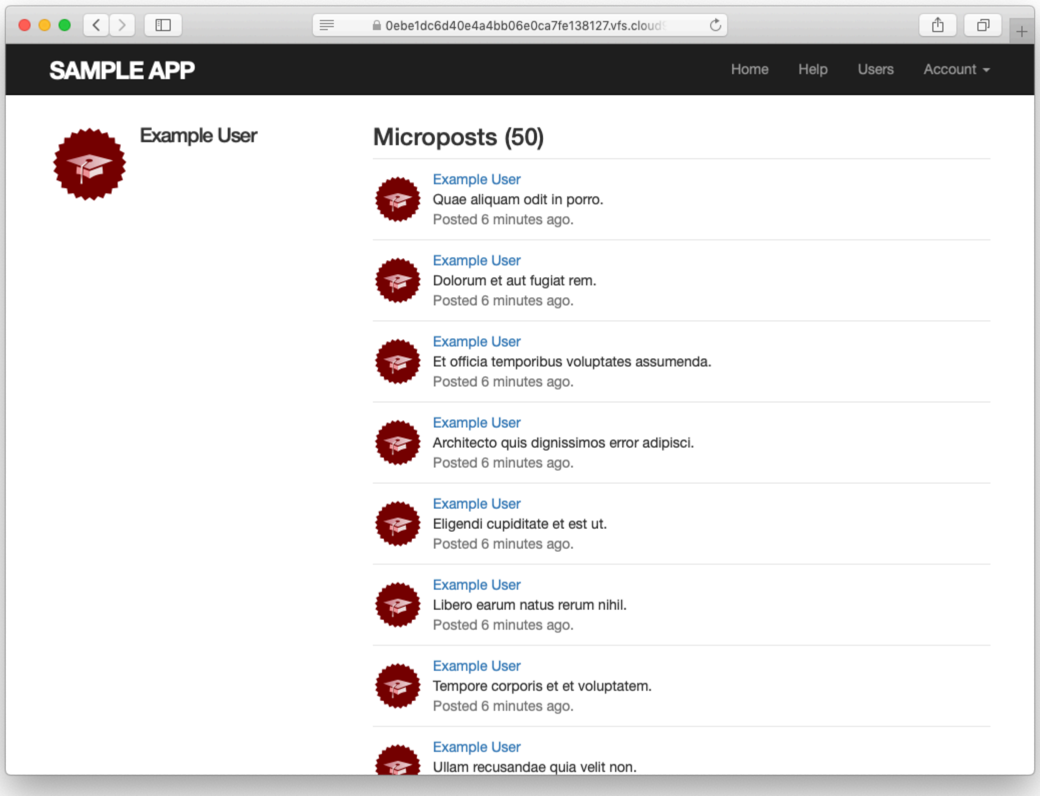
```
.
.
.
/* microposts */

.microposts {
  list-style: none;
  padding: 0;
  li {
    padding: 10px 0;
    border-top: 1px solid #e8e8e8;
  }
  .user {
    margin-top: 5em;
    padding-top: 0;
  }
  .content {
    display: block;
    margin-left: 60px;
    img {
      display: block;
      padding: 5px 0;
    }
  }
  .timestamp {
    color: $gray-light;
    display: block;
    margin-left: 60px;
  }
  .gravatar {
    float: left;
    margin-right: 10px;
    margin-top: 5px;
  }
}

aside {
  textarea {
    height: 100px;
    margin-bottom: 5px;
  }
}

span.image {
  margin-top: 10px;
  input {
    border: 0;
  }
}
```

Figure 13.7 shows the user profile page for the first user, while Figure 13.8



Example User

Microposts (50)

Example User
Quae aliquam odit in porro.
Posted 6 minutes ago.

Example User
Dolorum et aut fugiat rem.
Posted 6 minutes ago.

Example User
Et officia temporibus voluptates assumenda.
Posted 6 minutes ago.

Example User
Architecto quis dignissimos error adipisci.
Posted 6 minutes ago.

Example User
Eligendi cupiditate et est ut.
Posted 6 minutes ago.

Example User
Libero earum natus rerum nihil.
Posted 6 minutes ago.

Example User
Tempore corporis et et voluptatem.
Posted 6 minutes ago.

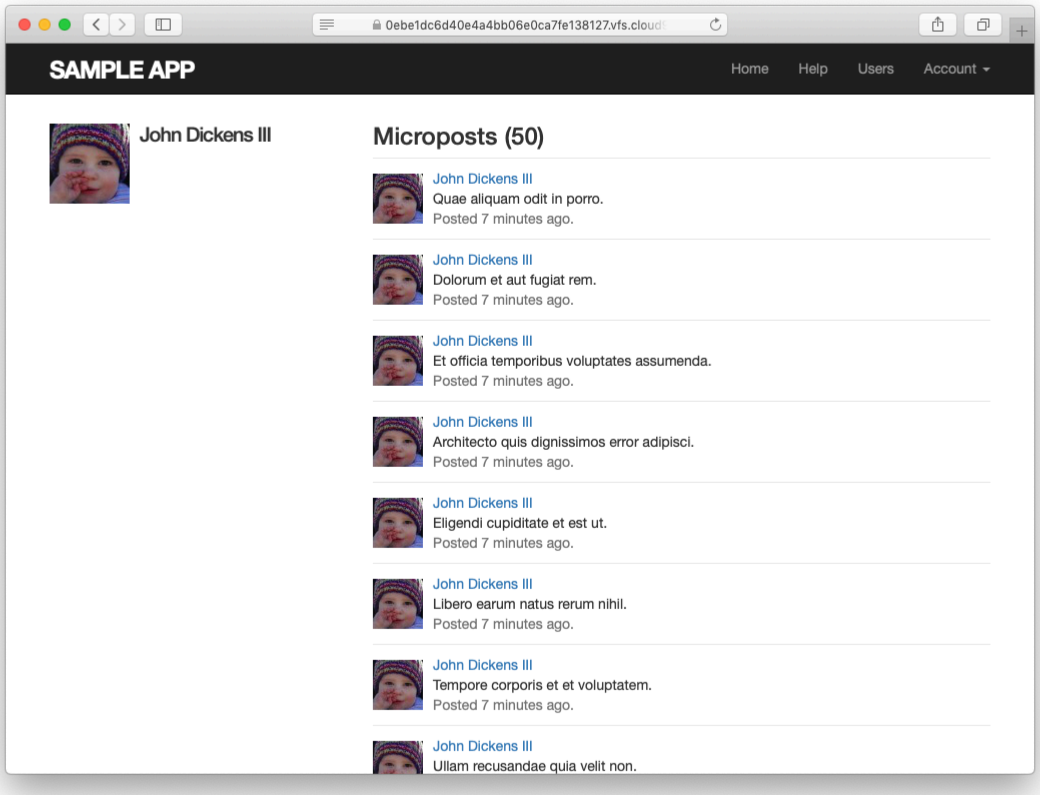
Example User
Ullam recusandae quia velit non.

Figure 13.7: The user profile with microposts (/users/1).

shows the profile for a second user. Finally, Figure 13.9 shows the *second* page of microposts for the first user, along with the pagination links at the bottom of the display. In all three cases, observe that each micropost display indicates the time since it was created (e.g., "Posted 1 minute ago."); this is the work of the **`time_ago_in_words`** method from Listing 13.22. If you wait a couple of minutes and reload the pages, you'll see how the text gets automatically updated based on the new time.

Exercises

Solutions to the exercises are available to all Rails Tutorial purchasers here.



John Dickens III

Microposts (50)

John Dickens III
Quae aliquam odit in porro.
Posted 7 minutes ago.

John Dickens III
Dolorum et aut fugiat rem.
Posted 7 minutes ago.

John Dickens III
Et officia temporibus voluptates assumenda.
Posted 7 minutes ago.

John Dickens III
Architecto quis dignissimos error adipisci.
Posted 7 minutes ago.

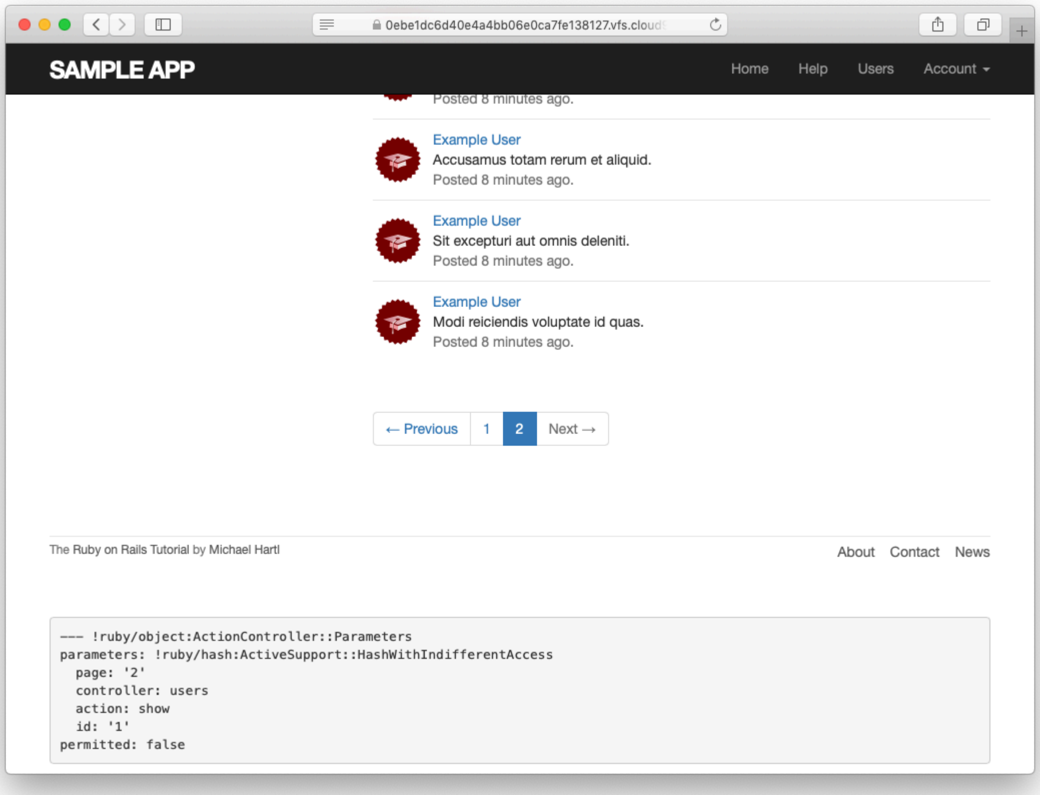
John Dickens III
Eligendi cupiditate et est ut.
Posted 7 minutes ago.

John Dickens III
Libero earum natus rerum nihil.
Posted 7 minutes ago.

John Dickens III
Tempore corporis et et voluptatem.
Posted 7 minutes ago.

John Dickens III
Ullam recusandae quia velit non.

Figure 13.8: The profile of a different user, also with microposts (/users/5).


0ebe1dc6d40e4a4bb06e0ca7fe138127.vfs.cloud
SAMPLE APP
Home
Help
Users
Account
Posted 8 minutes ago.
Example User
Accusamus totam rerum et aliquid.
Posted 8 minutes ago.
Example User
Sit excepturi aut omnis deleniti.
Posted 8 minutes ago.
Example User
Modi reiciendis voluptate id quas.
Posted 8 minutes ago.
← Previous
1
2
Next →
The Ruby on Rails Tutorial by Michael Hartl
About
Contact
News

```
--- !ruby/object:ActionController::Parameters
parameters: !ruby/hash:ActiveSupport::HashWithIndifferentAccess
  page: '2'
  controller: users
  action: show
  id: '1'
permitted: false
```

Figure 13.9: Micropost pagination links (/users/1?page=2).

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. See if you can guess the result of running **`(1..10).to_a.take(6)`**. Check at the console to see if your guess is right.

2. Is the **`to_a`** method in the previous exercise necessary?

3. Faker has a huge number of occasionally amusing applications. By consulting the Faker documentation, learn how to print out a fake university name, a fake phone number, a fake Hipster Ipsum sentence, and a fake Chuck Norris fact.

13.2.3 Profile micropost tests

Because newly activated users get redirected to their profile pages, we already have a test that the profile page renders correctly (Listing 11.33). In this section, we'll write a short integration test for some of the other elements on the profile page, including the work from this section. We'll start by generating an integration test for the profiles of our site's users:

```
$ rails generate integration_test users_profile
      invoke  test_unit
      create    test/integration/users_profile_test.rb
```

To test micropost pagination, we'll also generate some additional micropost fixtures using the same embedded Ruby technique we used to make additional users in Listing 10.47:

```
<% 30.times do |n| %>
micropost_<%= n %>:
  content: <%= Faker::Lorem.sentence(5) %>
  created_at: <%= 42.days.ago %>
  user: michael
<% end %>
```

Adding this to the code from Listing 13.15 gives the updated micropost fixtures in Listing 13.27.

Listing 13.27: Micropost fixtures with generated micropsts.
test/fixtures/microposts.yml

```
orange:
  content: "I just ate an orange!"
  created_at: <%= 10.minutes.ago %>
  user: michael

tau_manifesto:
  content: "Check out the @tauday site by @mhartl: https://tauday.com"
  created_at: <%= 3.years.ago %>
  user: michael

cat_video:
  content: "Sad cats are sad: https://youtu.be/PKffm2uI4dk"
  created_at: <%= 2.hours.ago %>
  user: michael

most_recent:
  content: "Writing a short test"
  created_at: <%= Time.zone.now %>
  user: michael

<% 30.times do |n| %>
micropost_<%= n %>:
  content: <%= Faker::Lorem.sentence(word_count: 5) %>
  created_at: <%= 42.days.ago %>
  user: michael
<% end %>
```

With the test data thus prepared, the test itself is fairly straightforward: we visit the user profile page and check for the page title and the user's name, Gravatar, micropost count, and paginated microposts. The result appears in Listing 13.28. Note the use of the **`full_title`** helper from Listing 4.2 to test the page's title, which we gain access to by including the Application Helper module into the test.[10]

[10]If you'd like to refactor other tests to use **`full_title`** (such as those in Listing 3.32), you should include the Application Helper in **`test_helper.rb`** instead.

Listing 13.28: A test for the user profile. 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_template 'users/show'
    assert_select 'title', full_title(@user.name)
    assert_select 'h1', text: @user.name
    assert_select 'h1>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
  end
end
```

The micropost count assertion in Listing 13.28 uses **`response.body`**, which we saw briefly in the Chapter 12 exercises (Section 12.3.3). Despite its name, **`response.body`** contains the full HTML source of the page (and not just the page's body). This means that if all we care about is that the number of microposts appears *somewhere* on the page, we can look for a match as follows:

```
assert_match @user.microposts.count.to_s, response.body
```

This is a much less specific assertion than **`assert_select`**; in particular, unlike **`assert_select`**, using **`assert_match`** in this context doesn't require us to indicate which HTML tag we're looking for.

Listing 13.28 also introduces the nesting syntax for **`assert_select`**:

```
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