7.1 Showing users

In this section, we'll take the first steps toward the final profile by making a page to display a user's name and profile photo, as indicated by the mockup in Figure 7.1.[1] Our eventual goal for the user profile pages is to show the user's profile image, basic user data, and a list of microposts, as mocked up in Figure 7.2.[2] (Figure 7.2 includes an example of *lorem ipsum* text, which has a fascinating story that you should definitely read about some time.) We'll complete this task, and with it the sample application, in Chapter 14.

If you're following along with version control, make a topic branch as usual:

```
$ git checkout -b sign-up
```

7.1.1 Debug and Rails environments

The profiles in this section will be the first truly dynamic pages in our application. Although the view will exist as a single page of code, each profile will be customized using information retrieved from the application's database. As preparation for adding dynamic pages to our sample application, now is a good time to add some debug information to our site layout (Listing 7.1). This displays some useful information about each page using the built-in **`debug`** method and **`params`** variable (which we'll learn more about in Section 7.1.2).

Listing 7.1: Adding some debug information to the site layout.
app/views/layouts/application.html.erb

```
<!DOCTYPE html>
<html>
  .
```

[1] Mockingbird doesn't support custom images like the profile photo in Figure 7.1; I put that in by hand using GIMP.

[2] Image retrieved from https://www.flickr.com/photos/43803060@N00/24308857/ on 2014-06-16. Copyright © 2002 by Shaun Wallin and used unaltered under the terms of the Creative Commons Attribution 2.0 Generic license.

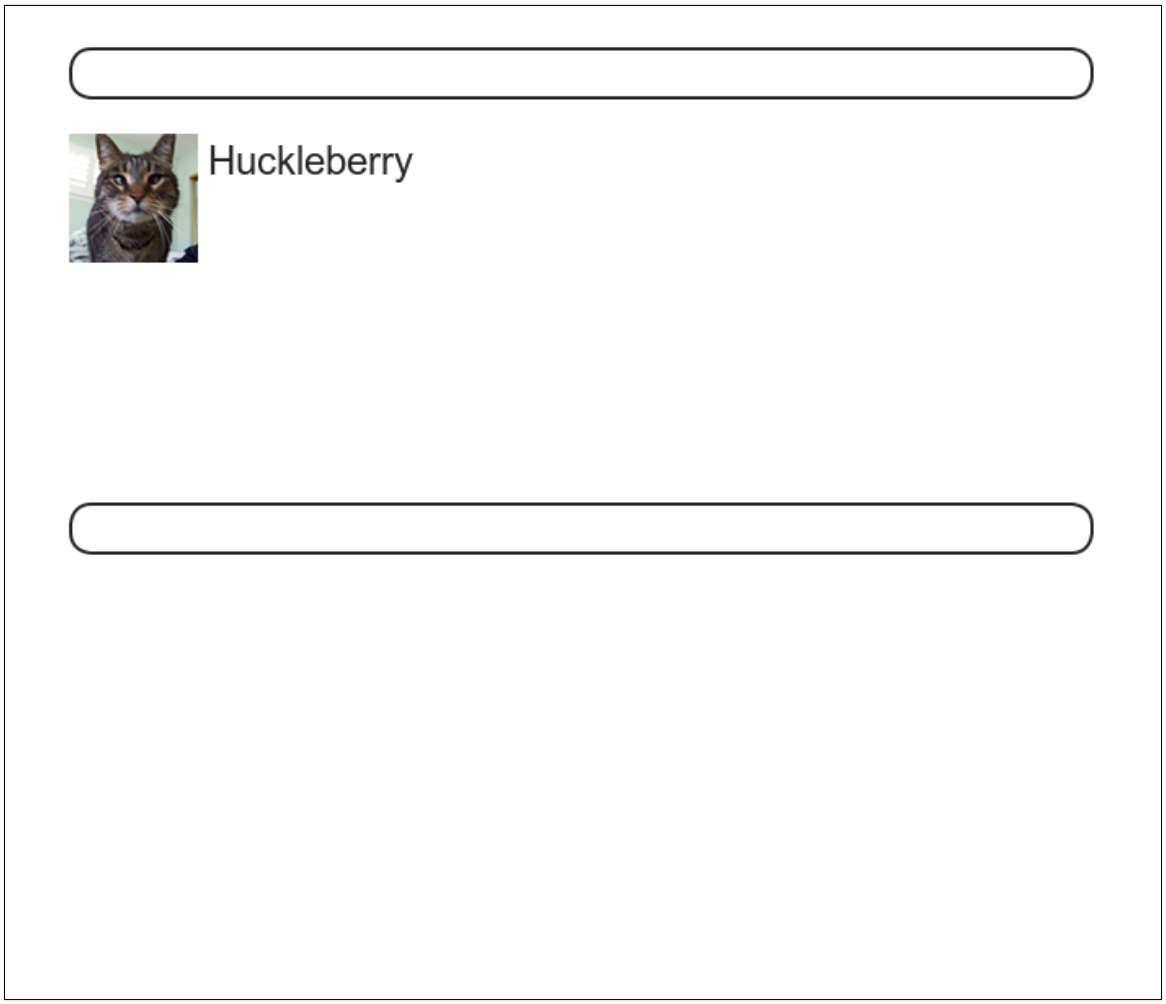

Figure 7.1: A mockup of the user profile made in this section.

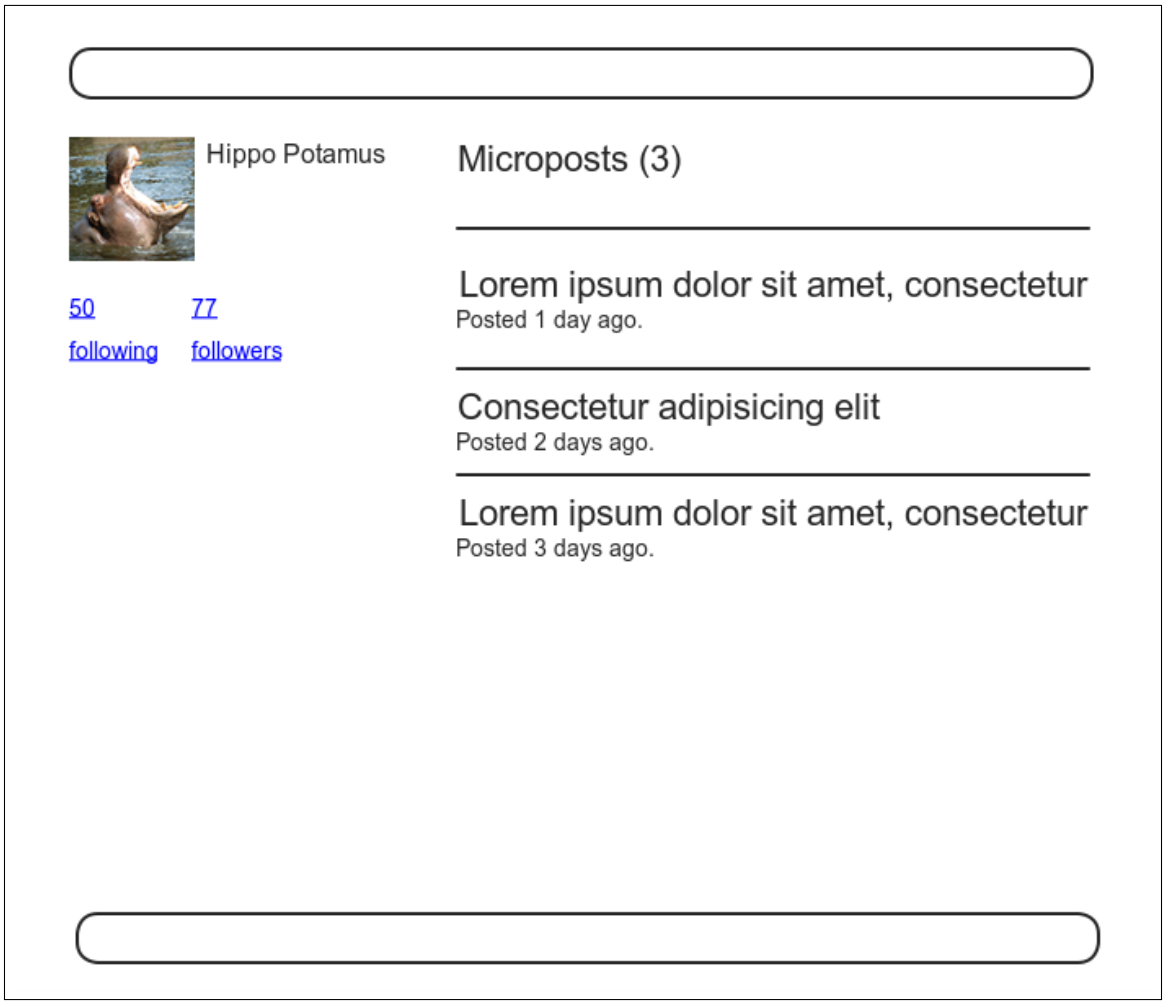

Figure 7.2: A mockup of our best guess at the final profile page.

```
  .
  .
  <body>
    <%= render 'layouts/header' %>
    <div class="container">
      <%= yield %>
      <%= render 'layouts/footer' %>
      <%= debug(params) if Rails.env.development? %>
    </div>
  </body>
</html>
```

Since we don't want to display debug information to users of a deployed application, Listing 7.1 uses

```
if Rails.env.development?
```

to restrict the debug information to the *development environment*, which is one of three environments defined by default in Rails (Box 7.1).[3] In particular, **`Rails.env.development?`** is **`true`** only in a development environment, so the embedded Ruby

```
<%= debug(params) if Rails.env.development? %>
```

won't be inserted into production applications or tests. (Inserting the debug information into tests probably wouldn't do any harm, but it probably wouldn't do any good, either, so it's best to restrict the debug display to development only.)

Box 7.1. Rails environments

Rails comes equipped with three environments: `test`, `development`, and `production`. The default environment for the Rails console is `development`:

[3] You can define your own custom environments as well; see the RailsCast on adding an environment for details.

```
$ rails console
Loading development environment
>> Rails.env
=> "development"
>> Rails.env.development?
=> true
>> Rails.env.test?
=> false
```

As you can see, Rails provides a `Rails` object with an `env` attribute and associated environment boolean methods, so that, for example, `Rails.env.test?` returns `true` in a test environment and `false` otherwise.

If you ever need to run a console in a different environment (to debug a test, for example), you can pass the environment as a parameter to the `console` script:

```
$ rails console test
Loading test environment
>> Rails.env
=> "test"
>> Rails.env.test?
=> true
```

As with the console, `development` is the default environment for the Rails server, but you can also run it in a different environment:

```
$ rails server --environment production
```

If you view your app running in production, it won't work without a production database, which we can create by running `rails db:migrate` in production:

```
$ rails db:migrate RAILS_ENV=production
```

(I find it confusing that the idiomatic commands to run the console, server, and migrate commands in non-default environments use different syntax, which is why I bothered showing all three. It's worth noting, though, that preceding any of them with `RAILS_ENV=<env>` will also work, as in `RAILS_ENV=production rails server`).

By the way, if you have deployed your sample app to Heroku, you can see its environment using `heroku run rails console`:

```
$ heroku run rails console
>> Rails.env
=> "production"
>> Rails.env.production?
=> true
```

Naturally, since Heroku is a platform for production sites, it runs each application in a production environment.

To make the debug output look nice, we'll add some rules to the custom stylesheet created in Chapter 5, as shown in Listing 7.2.

Listing 7.2: Adding code for a pretty debug box, including a Sass mixin.
app/assets/stylesheets/custom.scss

```
@import "bootstrap-sprockets";
@import "bootstrap";

/* mixins, variables, etc. */

$gray-medium-light: #eaeaea;

@mixin box_sizing {
  -moz-box-sizing:    border-box;
  -webkit-box-sizing: border-box;
  box-sizing:         border-box;
}
.
.
```

```
.
/* miscellaneous */

.debug_dump {
  clear: both;
  float: left;
  width: 100%;
  margin-top: 45px;
  @include box_sizing;
}
```

This introduces the Sass *mixin* facility, in this case called **box_sizing**. A mixin allows a group of CSS rules to be packaged up and used for multiple elements, converting

```
.debug_dump {
  .
  .
  .
  @include box_sizing;
}
```

to

```
.debug_dump {
  .
  .
  .
  -moz-box-sizing:    border-box;
  -webkit-box-sizing: border-box;
  box-sizing:         border-box;
}
```

We'll put this mixin to use again in Section 7.2.1. The result in the case of the debug box is shown in Figure 7.3.[4]

The debug output in Figure 7.3 gives potentially useful information about the page being rendered:

[4]The exact appearance of the Rails debug information is slightly version-dependent. For example, as of Rails 5 the debug information shows the **permitted** status of the information, a subject we'll cover in Section 7.3.2. Use your technical sophistication (Box 1.2) to resolve such minor discrepancies.

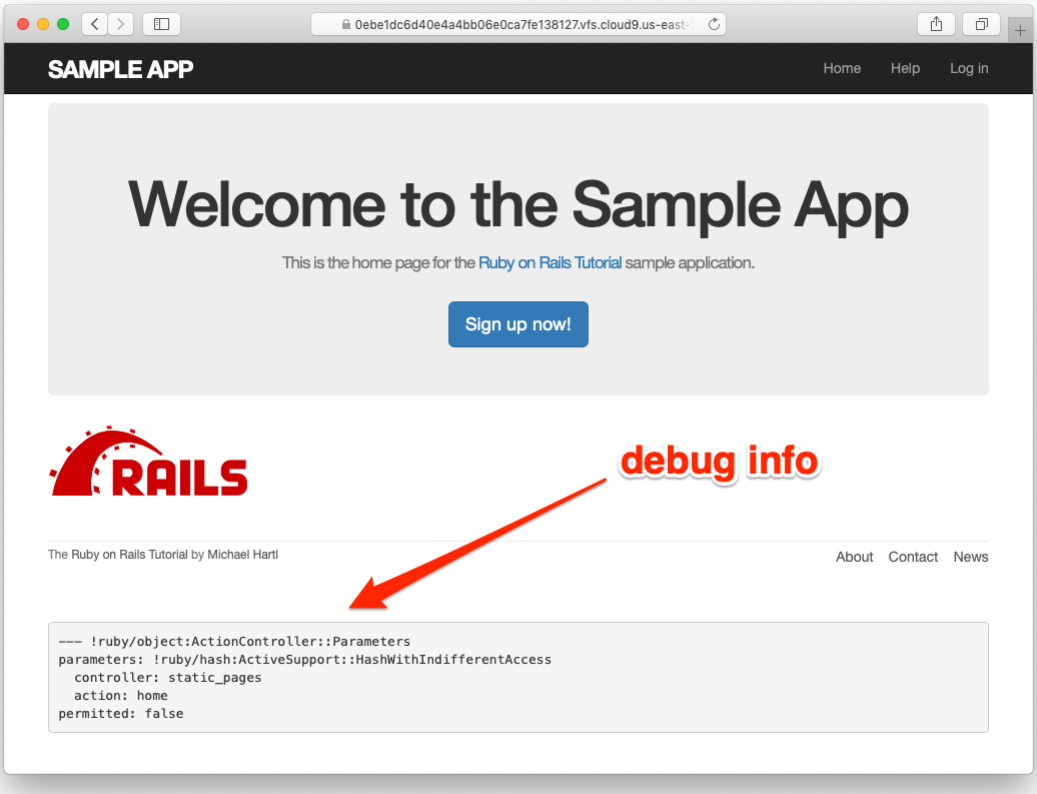

Figure 7.3: The sample application Home page with debug information.

```
---
controller: static_pages
action: home
```

This is a YAML[5] representation of `params`, which is basically a hash, and in this case identifies the controller and action for the page. We'll see another example in Section 7.1.2.

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. Visit /about in your browser and use the debug information to determine the controller and action of the `params` hash.

2. In the Rails console, pull the first user out of the database and assign it to the variable `user`. What is the output of `puts user.attributes.-to_yaml`? Compare this to using the `y` method via `y user.attributes`.

7.1.2 A Users resource

In order to make a user profile page, we need to have a user in the database, which introduces a chicken-and-egg problem: how can the site have a user before there is a working signup page? Happily, this problem has already been solved: in Section 6.3.4, we created a User record by hand using the Rails console, so there should be one user in the database:

[5] The Rails `debug` information is shown as YAML (a recursive acronym standing for "YAML Ain't Markup Language"), which is a friendly data format designed to be both machine- *and* human-readable.