```
---
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.

```
$ rails console
>> User.count
=> 1
>> User.first
=> #<User id: 1, name: "Michael Hartl", email: "mhartl@example.com",
created_at: "2019-08-22 03:15:38", updated_at: "2019-08-22 03:15:38",
password_digest: [FILTERED]>
```

(If you don't currently have a user in your database, you should visit Section 6.3.4 now and complete it before proceeding.) We see from the console output above that the user has id `1`, and our goal now is to make a page to display this user's information. We'll follow the conventions of the REST architecture favored in Rails applications (Box 2.2), which means representing data as *resources* that can be created, shown, updated, or destroyed—four actions corresponding to the four fundamental operations `POST`, `GET`, `PATCH`, and `DELETE` defined by the HTTP standard (Box 3.2).

When following REST principles, resources are typically referenced using the resource name and a unique identifier. What this means in the context of users—which we're now thinking of as a Users *resource*—is that we should view the user with id `1` by issuing a `GET` request to the URL /users/1. Here the **`show`** action is *implicit* in the type of request—when Rails' REST features are activated, `GET` requests are automatically handled by the **`show`** action.

We saw in Section 2.2.1 that the page for a user with id `1` has URL /users/1. Unfortunately, visiting that URL right now just gives an error (Figure 7.4).

We can get the routing for /users/1 to work by adding a single line to our routes file (**`config/routes.rb`**):

```
resources :users
```

The result appears in Listing 7.3.

Listing 7.3: Adding a Users resource to the routes file.
`config/routes.rb`

```
Rails.application.routes.draw do
  root 'static_pages#home'
```

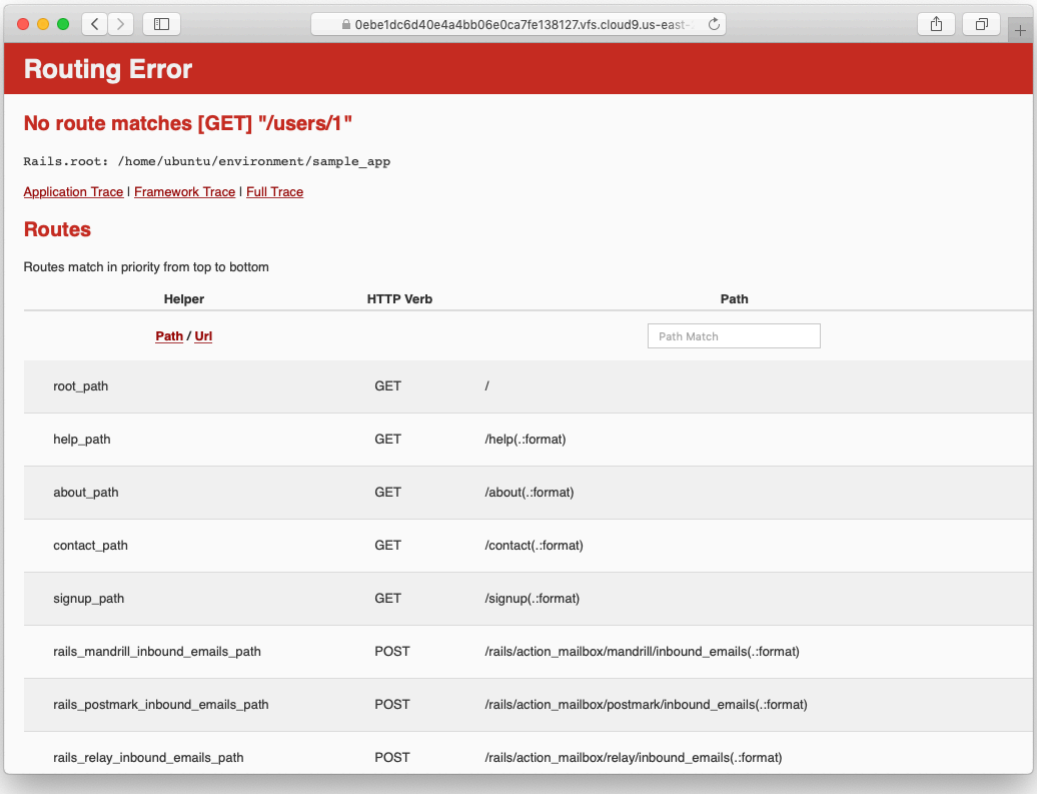

Figure 7.4: The current state of /users/1.

HTTP request	URL	Action	Named route	Purpose
GET	/users	index	users_path	page to list all users
GET	/users/1	show	user_path(user)	page to show user
GET	/users/new	new	new_user_path	page to make a new user (signup)
POST	/users	create	users_path	create a new user
GET	/users/1/edit	edit	edit_user_path(user)	page to edit user with id 1
PATCH	/users/1	update	user_path(user)	update user
DELETE	/users/1	destroy	user_path(user)	delete user

Table 7.1: RESTful routes provided by the Users resource in Listing 7.3.

```
  get  '/help',    to: 'static_pages#help'
  get  '/about',   to: 'static_pages#about'
  get  '/contact', to: 'static_pages#contact'
  get  '/signup',  to: 'users#new'
  resources :users
end
```

Although our immediate motivation is making a page to show users, the single line **`resources :users`** doesn't just add a working /users/1 URL; it endows our sample application with *all* the actions needed for a RESTful Users resource,[6] along with a large number of named routes (Section 5.3.3) for generating user URLs. The resulting correspondence of URLs, actions, and named routes is shown in Table 7.1. (Compare to Table 2.2.) Over the course of the next three chapters, we'll cover all of the other entries in Table 7.1 as we fill in all the actions necessary to make Users a fully RESTful resource.

With the code in Listing 7.3, the routing works, but there's still no page there (Figure 7.5). To fix this, we'll begin with a minimalist version of the profile page, which we'll flesh out in Section 7.1.4.

We'll use the standard Rails location for showing a user, which is **`app/views/users/show.html.erb`**. Unlike the **`new.html.erb`** view, which we created with the generator in Listing 5.38, the **`show.html.erb`** file doesn't currently exist, so you'll have to create it by hand,[7] and then fill it with the

[6] This means that the *routing* works, but the corresponding pages don't necessarily work at this point. For example, /users/1/edit gets routed properly to the **`edit`** action of the Users controller, but since the **`edit`** action doesn't exist yet actually hitting that URL will return an error.

[7] Using, e.g., **`touch app/views/users/show.html.erb`**.

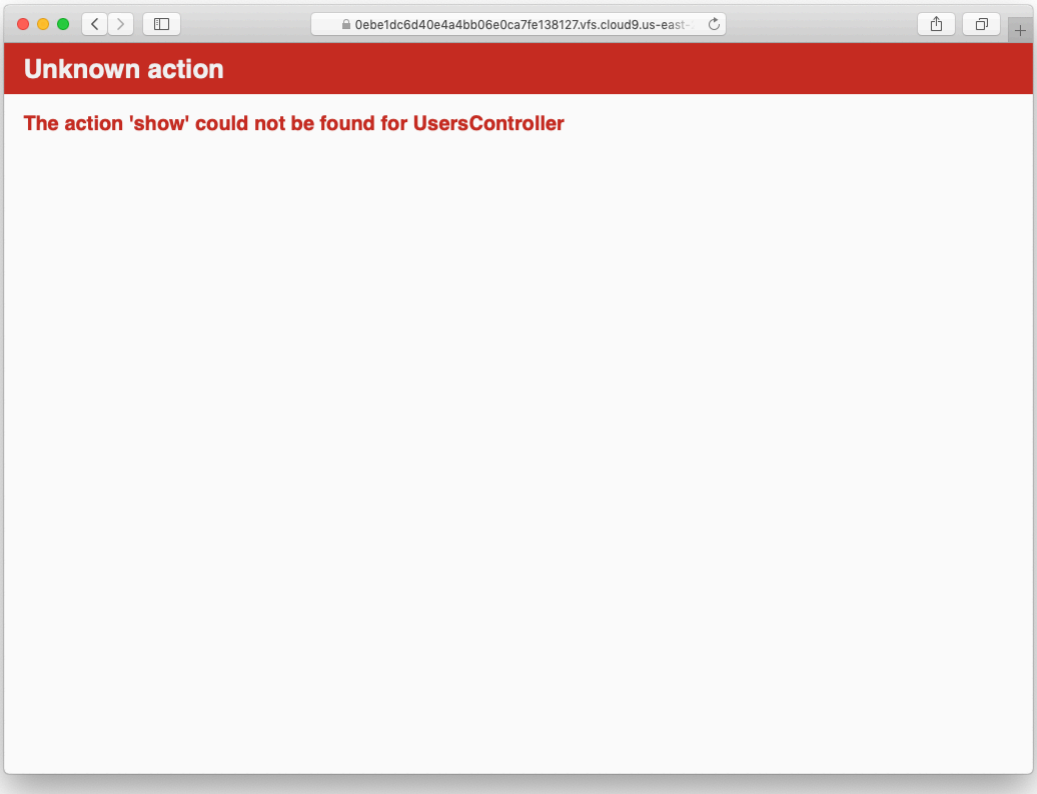

Figure 7.5: The URL /users/1 with routing but no page.

content shown in Listing 7.4.

Listing 7.4: A stub view for showing user information.
app/views/users/show.html.erb

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

This view uses embedded Ruby to display the user's name and email address, assuming the existence of an instance variable called **@user**. Of course, eventually the real user show page will look very different (and won't display the email address publicly).

In order to get the user show view to work, we need to define an **@user** variable in the corresponding **show** action in the Users controller. As you might expect, we use the **find** method on the User model (Section 6.1.4) to retrieve the user from the database, as shown in Listing 7.5.

Listing 7.5: The Users controller with a **show** action.
app/controllers/users_controller.rb

```
class UsersController < ApplicationController

  def show
    @user = User.find(params[:id])
  end

  def new
  end
end
```

Here we've used **params** to retrieve the user id. When we make the appropriate request to the Users controller, **params[:id]** will be the user id 1, so the effect is the same as the **find** method **User.find(1)** we saw in Section 6.1.4. (Technically, **params[:id]** is the string **"1"**, but **find** is smart enough to convert this to an integer.)

With the user view and action defined, the URL /users/1 works perfectly, as seen in Figure 7.6. (If you haven't restarted the Rails server since adding

bcrypt, you may have to do so at this time. This sort of thing is a good application of technical sophistication (Box 1.2).) Note that the debug information in Figure 7.6 confirms the value of **`params[:id]`**:

```
---
action: show
controller: users
id: '1'
```

This is why the code

```
User.find(params[:id])
```

in Listing 7.5 finds the user with id `1`.

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. Using embedded Ruby, add the **`created_at`** and **`updated_at`** "magic column" attributes to the user show page from Listing 7.4.
2. Using embedded Ruby, add **`Time.now`** to the user show page. What happens when you refresh the browser?

7.1.3 Debugger

We saw in Section 7.1.2 how the information in the **`debug`** could help us understand what's going on in our application, but there's also a more direct way to get debugging information using the `byebug` gem (Listing 3.2). To see how it works, we just need to add a line consisting of **`debugger`** to our application, as shown in Listing 7.6.