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.

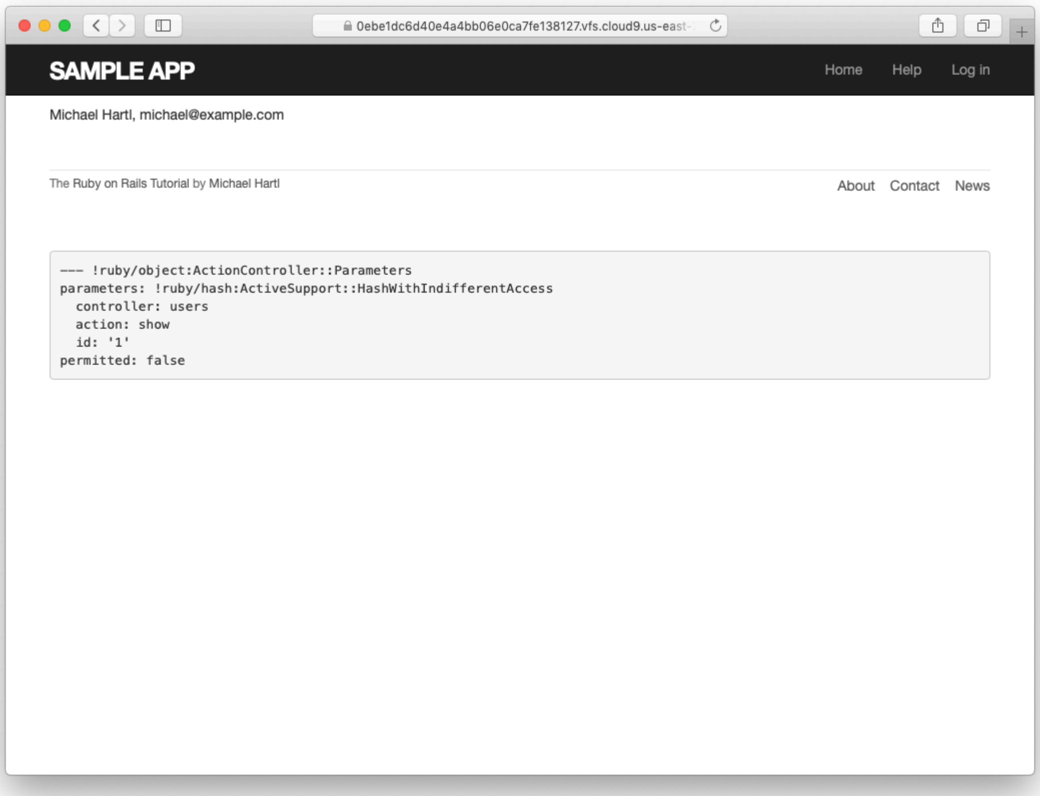
To see other people's answers and to record your own, subscribe to the Rails Tutorial course or to the Learn Enough All Access Bundle.

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.

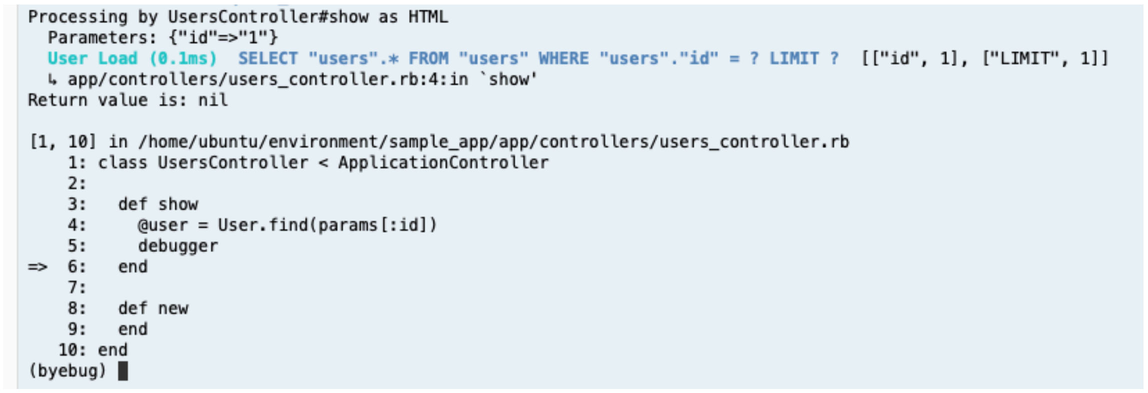


Michael Hartl, michael@example.com

The Ruby on Rails Tutorial by Michael Hartl

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

Figure 7.6: The user show page after adding a Users resource.



```
Processing by UsersController#show as HTML
  Parameters: {"id"=>"1"}
  User Load (0.1ms)  SELECT "users".* FROM "users" WHERE "users"."id" = ? LIMIT ?  [["id", 1], ["LIMIT", 1]]
  ↳ app/controllers/users_controller.rb:4:in `show'
Return value is: nil

[1, 10] in /home/ubuntu/environment/sample_app/app/controllers/users_controller.rb
    1: class UsersController < ApplicationController
    2:
    3:   def show
    4:     @user = User.find(params[:id])
    5:     debugger
=>  6:   end
    7:
    8:   def new
    9:   end
   10: end
(byebug) █
```

Figure 7.7: The **byebug** prompt in the Rails server.

Listing 7.6: The Users controller with a debugger.
app/controllers/users_controller.rb

```
class UsersController < ApplicationController

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

  def new
  end
end
```

Now, when we visit /users/1, the Rails server shows a **byebug** prompt (Figure 7.7):

```
(byebug)
```

We can treat **byebug** like a Rails console, issuing commands to figure out the state of the application:

```
(byebug) @user.name
"Michael Hartl"
(byebug) @user.email
"michael@example.com"
(byebug) params[:id]
"1"
```

To release the prompt and continue execution of the application, press Ctrl-D, then remove the **`debugger`** line from the **`show`** action (Listing 7.7).

Listing 7.7: The Users controller with the debugger line removed.
`app/controllers/users_controller.rb`

```ruby
class UsersController < ApplicationController

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

  def new
  end
end
```

Whenever you're confused about something in a Rails application, it's a good practice to put **`debugger`** close to the code you think might be causing the trouble. Inspecting the state of the system using `byebug` is a powerful method for tracking down application errors and interactively debugging your application.

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. With the **`debugger`** in the **`show`** action as in Listing 7.6, hit /users/1. Use **`puts`** to display the value of the YAML form of the **`params`** hash. *Hint*: Refer to the relevant exercise in Section 7.1.1. How does it compare to the debug information shown by the **`debug`** method in the site template?