Listing 9.20: 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 fix in Listing 9.16 and then verify that the first subtle bug is present by opening two logged-in tabs, logging out in one, and then clicking "Log out" link in the other.

2. Comment out the fix in Listing 9.19 and verify that the second subtle bug is present by logging out in one browser and closing and opening the second browser.

3. Uncomment the fixes and confirm that the test suite goes from RED to GREEN.

9.2 "Remember me" checkbox

With the code in Section 9.1.3, our application has a complete, professional-grade authentication system. As a final step, we'll see how to make staying logged in optional using a "remember me" checkbox. A mockup of the login form with such a checkbox appears in Figure 9.3.

To write the implementation, we start by adding a checkbox to the login form from Listing 8.4. As with labels, text fields, password fields, and submit buttons, checkboxes can be created with a Rails helper method. In order to get the styling right, though, we have to *nest* the checkbox inside the label, as follows:

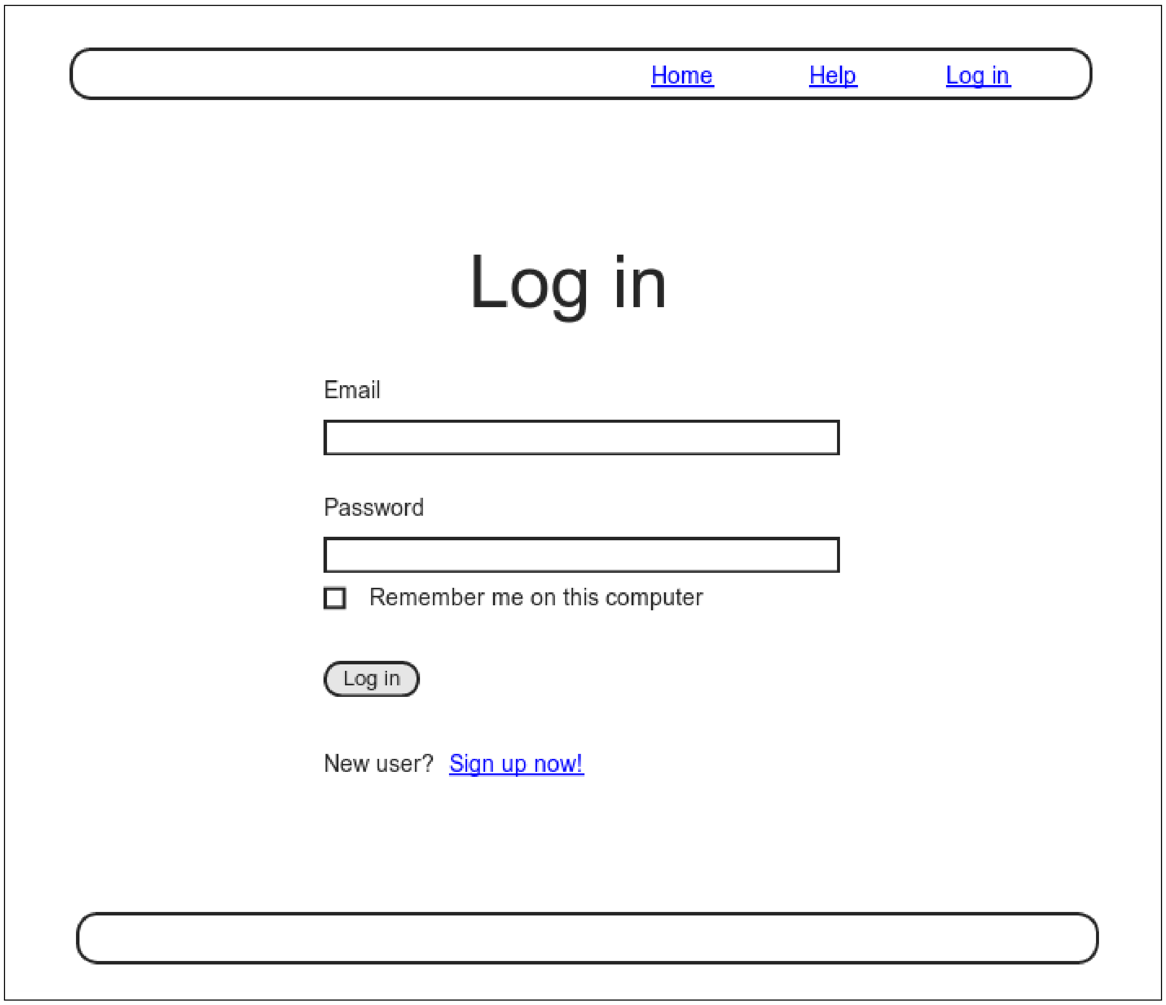


Figure 9.3: A mockup of a “remember me” checkbox.

```
<%= f.label :remember_me, class: "checkbox inline" do %>
  <%= f.check_box :remember_me %>
  <span>Remember me on this computer</span>
<% end %>
```

Putting this into the login form gives the code shown in Listing 9.21.

Listing 9.21: Adding a "remember me" checkbox to the login form.
app/views/sessions/new.html.erb

```
<% provide(:title, "Log in") %>
<h1>Log in</h1>

<div class="row">
  <div class="col-md-6 col-md-offset-3">
    <%= form_with(url: login_path, scope: :session, local: true) do |f| %>

      <%= f.label :email %>
      <%= f.email_field :email, class: 'form-control' %>

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

      <%= f.label :remember_me, class: "checkbox inline" do %>
        <%= f.check_box :remember_me %>
        <span>Remember me on this computer</span>
      <% end %>

      <%= f.submit "Log in", class: "btn btn-primary" %>
    <% end %>

    <p>New user? <%= link_to "Sign up now!", signup_path %></p>
  </div>
</div>
```

In Listing 9.21, we've included the CSS classes **`checkbox`** and **`inline`**, which Bootstrap uses to put the checkbox and the text ("Remember me on this computer") in the same line. In order to complete the styling, we need just a few more CSS rules, as shown in Listing 9.22. The resulting login form appears in Figure 9.4.

Listing 9.22: CSS for the “remember me” checkbox.
app/assets/stylesheets/custom.scss

```
.
.
.
/* forms */
.
.
.
.checkbox {
  margin-top: -10px;
  margin-bottom: 10px;
  span {
    margin-left: 20px;
    font-weight: normal;
  }
}

#session_remember_me {
  width: auto;
  margin-left: 0;
}
```

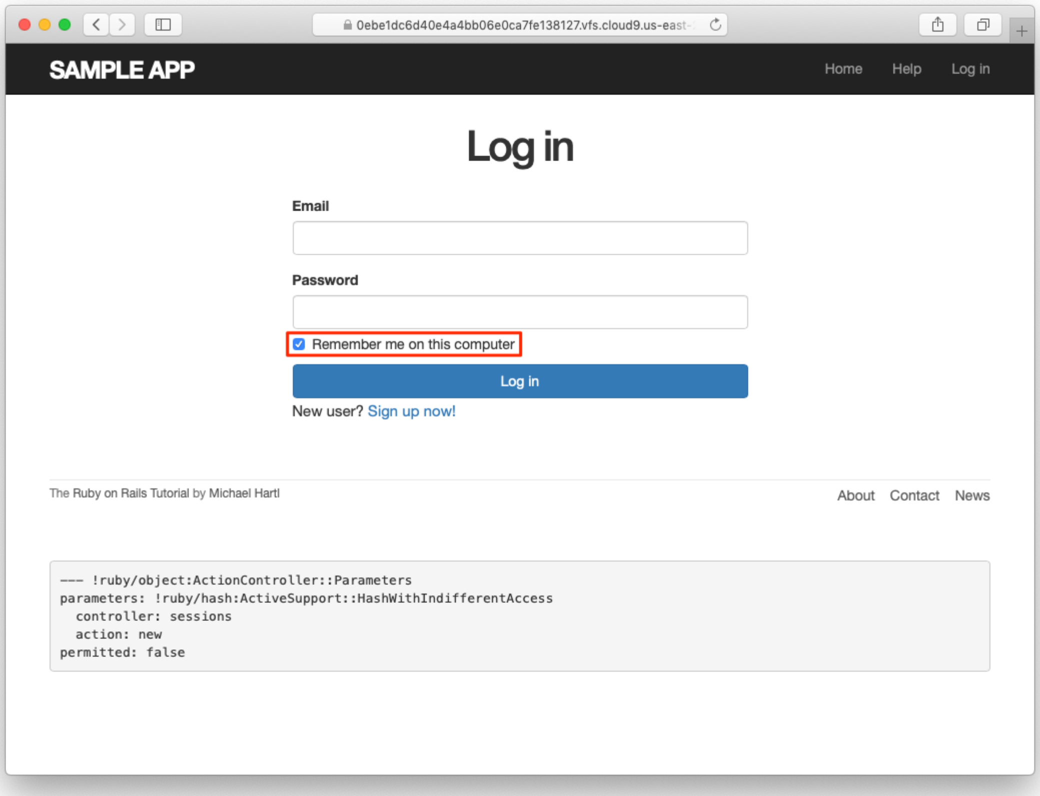
Having edited the login form, we’re now ready to remember users if they check the checkbox and forget them otherwise. Incredibly, because of all our work in the previous sections, the implementation can be reduced to one line. We start by noting that the `params` hash for submitted login forms now includes a value based on the checkbox (as you can verify by submitting the form in Listing 9.21 with invalid information and inspecting the values in the debug section of the page). In particular, the value of

```
params[:session][:remember_me]
```

is `'1'` if the box is checked and `'0'` if it isn’t.

By testing the relevant value of the `params` hash, we can now remember or forget the user based on the value of the submission:[15]

[15]Note that this means unchecking the box will log out the user on all browsers on all computers. The alternate design of remembering user login sessions on each browser independently is potentially more convenient for users, but it’s less secure, and is also more complicated to implement. Ambitious readers are invited to try their hand at implementing it.


0ebe1dc6d40e4a4bb06e0ca7fe138127.vfs.cloud9.us-east-
SAMPLE APP
Home
Help
Log in
Log in
Email
Password
Remember me on this computer
Log in
New user? Sign up now!
The Ruby on Rails Tutorial by Michael Hartl
About
Contact
News

```
--- !ruby/object:ActionController::Parameters
parameters: !ruby/hash:ActiveSupport::HashWithIndifferentAccess
  controller: sessions
  action: new
permitted: false
```

Figure 9.4: The login form with an added “remember me” checkbox.

```
if params[:session][:remember_me] == '1'
  remember(user)
else
  forget(user)
end
```

As explained in Box 9.2, this sort of **`if-then`** branching structure can be converted to one line using the *ternary operator* as follows:[16]

```
params[:session][:remember_me] == '1' ? remember(user) : forget(user)
```

Using this to replace **`remember user`** in the Sessions controller's **`create`** method (Listing 9.7) leads to the amazingly compact code shown in Listing 9.23. (Now you're in a position to understand the code in Listing 8.22, which uses the ternary operator to define the bcrypt **`cost`** variable.)

Listing 9.23: Handling the submission of the "remember me" checkbox.
app/controllers/sessions_controller.rb

```
class SessionsController < ApplicationController

  def new
  end

  def create
    user = User.find_by(email: params[:session][:email].downcase)
    if user && user.authenticate(params[:session][:password])
      log_in user
      params[:session][:remember_me] == '1' ? remember(user) : forget(user)
      redirect_to user
    else
      flash.now[:danger] = 'Invalid email/password combination'
      render 'new'
    end
  end

  def destroy
    log_out if logged_in?
    redirect_to root_url
  end
end
```

[16]Before we wrote **`remember user`** without parentheses, but when used with the ternary operator omitting them results in a syntax error.

With the implementation in Listing 9.23, our login system is complete, as you can verify by checking or unchecking the box in your browser.

Box 9.2. 10 types of people

There’s an old joke that there are 10 kinds of people in the world: those who understand binary and those who don’t (10, of course, being 2 in binary). In this spirit, we can say that there are 10 kinds of people in the world: those who like the ternary operator, those who don’t, and those who don’t yet know about it. (If you happen to be in the third category, soon you won’t be any longer.)

When you do a lot of programming, you quickly learn that one of the most common bits of control flow goes something like this:

```
if boolean?
  do_one_thing
else
  do_something_else
end
```

Ruby, like many other languages (including C/C++, Perl, PHP, and Java), allows you to replace this with a much more compact expression using the *ternary operator* (so called because it consists of three parts):

```
boolean? ? do_one_thing : do_something_else
```

You can also use the ternary operator to replace assignment, so that

```
if boolean?
  var = foo
else
  var = bar
end
```

becomes

```
  var = boolean? ? foo : bar
```

Finally, it's often convenient to use the ternary operator in a function's return value:

```
  def foo
    do_stuff
    boolean? ? "bar" : "baz"
  end
```

Since Ruby implicitly returns the value of the last expression in a function, here the `foo` method returns `"bar"` or `"baz"` depending on whether `boolean?` is `true` or `false`.

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. By inspecting your browser's cookies directly, verify that the "remember me" checkbox is having its intended effect.

2. At the console, invent examples showing both possible behaviors of the ternary operator (Box 9.2).

9.3 Remember tests

Although our "remember me" functionality is now working, it's important to write some tests to verify its behavior. One reason is to catch implementation errors, as discussed in a moment. Even more important, though, is that the core user persistence code is in fact completely untested at present. Fixing these