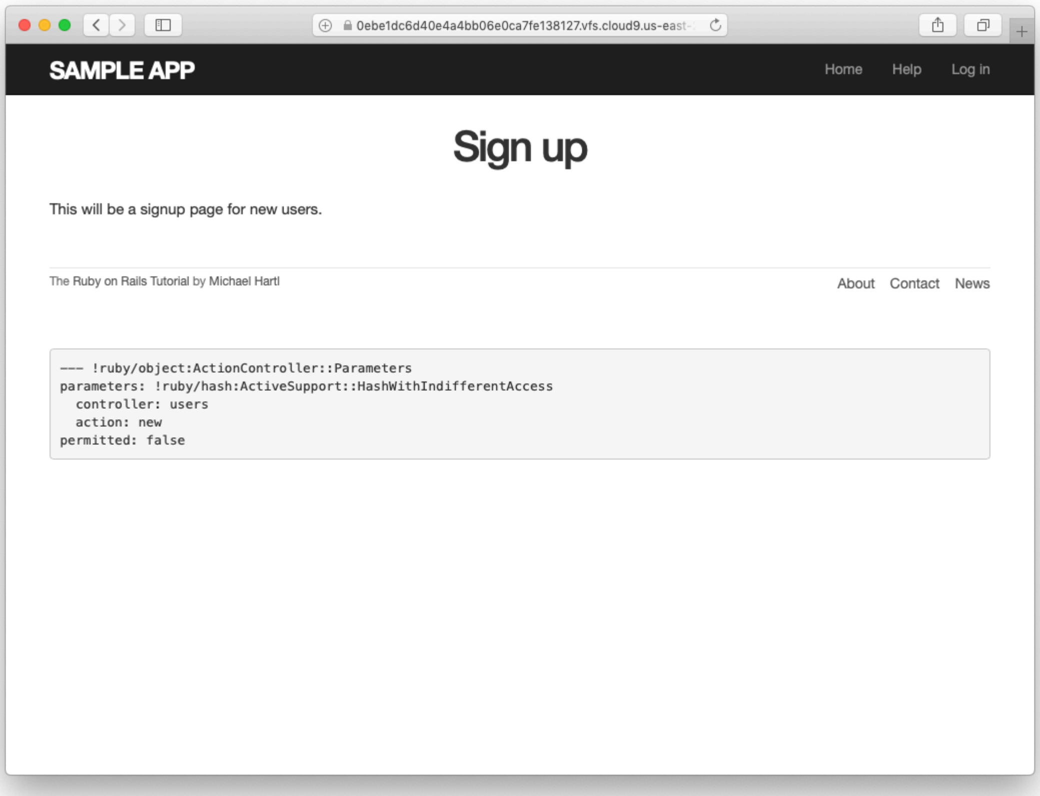


Sign up

This will be a signup page for new users.

The Ruby on Rails Tutorial by Michael Hartl

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

Figure 7.11: The current state of the signup page /signup.

7.2 Signup form

Now that we have a working (though not yet complete) user profile page, we're ready to make a signup form for our site. We saw in Figure 5.11 (shown again in Figure 7.11) that the signup page is currently blank: useless for signing up new users. The goal of this section is to start changing this sad state of affairs by producing the signup form mocked up in Figure 7.12.

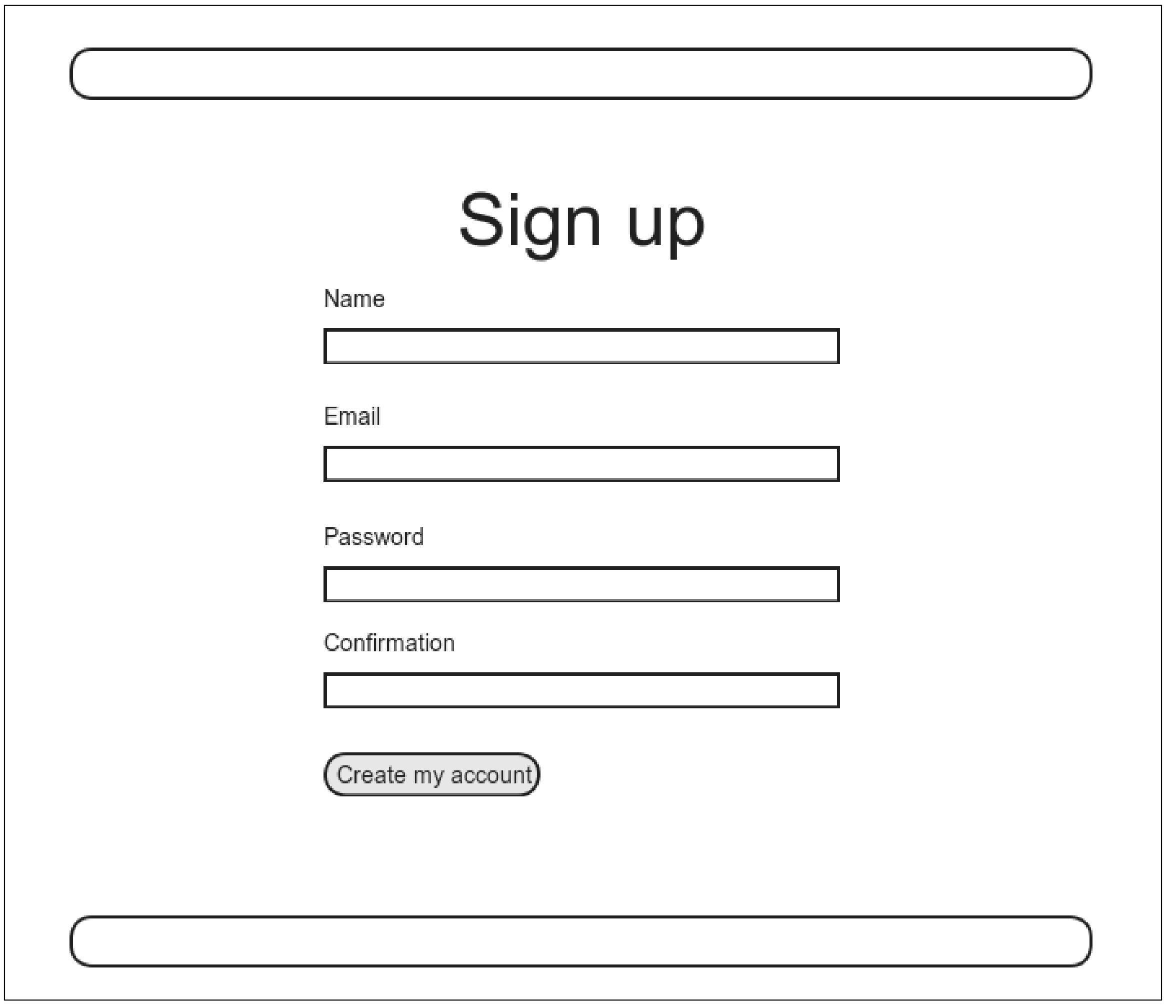


Figure 7.12: A mockup of the user signup page.

7.2.1 Using `form_with`

The heart of the signup page is a *form* for submitting the relevant signup information (name, email, password, confirmation). We can accomplish this in Rails with the **`form_with`** helper method, which uses an Active Record object to build a form using the object's attributes.

Recalling that the signup page /signup is routed to the **`new`** action in the Users controller (Listing 5.43), our first step is to create the User object required as an argument to **`form_with`**. The resulting **`@user`** variable definition appears in Listing 7.14.

Listing 7.14: Adding an **`@user`** variable to the **`new`** action.
`app/controllers/users_controller.rb`

```ruby
class UsersController < ApplicationController

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

  def new
    @user = User.new
  end
end
```

The form itself appears as in Listing 7.15. We'll discuss it in detail in Section 7.2.2, but first let's style it a little with the SCSS in Listing 7.16. (Note the reuse of the **`box_sizing`** mixin from Listing 7.2.) Once these CSS rules have been applied, the signup page appears as in Figure 7.13.

Listing 7.15: A form to sign up new users.
`app/views/users/new.html.erb`

```erb
<% provide(:title, 'Sign up') %>
<h1>Sign up</h1>

<div class="row">
  <div class="col-md-6 col-md-offset-3">
    <%= form_with(model: @user, local: true) do |f| %>
      <%= f.label :name %>
      <%= f.text_field :name %>

      <%= f.label :email %>
      <%= f.email_field :email %>

      <%= f.label :password %>
      <%= f.password_field :password %>

      <%= f.label :password_confirmation, "Confirmation" %>
      <%= f.password_field :password_confirmation %>

      <%= f.submit "Create my account", class: "btn btn-primary" %>
    <% end %>
  </div>
</div>
```

Listing 7.16: CSS for the signup form.
app/assets/stylesheets/custom.scss

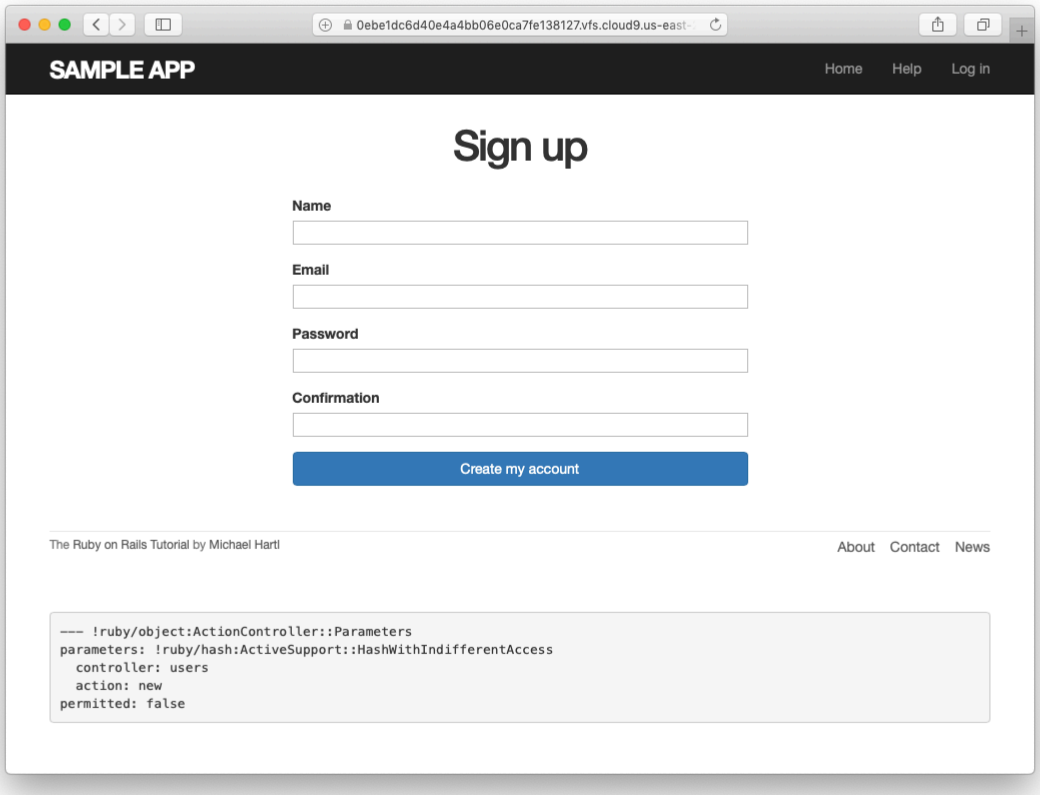
```
.
.
.
/* forms */

input, textarea, select, .uneditable-input {
  border: 1px solid #bbb;
  width: 100%;
  margin-bottom: 15px;
  @include box_sizing;
}

input {
  height: auto !important;
}
```

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. Confirm by replacing all occurrences of `f` with `foobar` that the name of the block variable is irrelevant as far as the result is concerned. Why might `foobar` nevertheless be a bad choice?


0ebe1dc6d40e4a4bb06e0ca7fe138127.vfs.cloud9.us-east-
SAMPLE APP
Home
Help
Log in
Sign up
Name
Email
Password
Confirmation
Create my account
The Ruby on Rails Tutorial by Michael Hartl
About
Contact
News

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

Figure 7.13: The user signup form.

7.2.2 Signup form HTML

To understand the form defined in Listing 7.15, it's helpful to break it into smaller pieces. We'll first look at the outer structure, which consists of embedded Ruby opening with a call to **`form_with`** and closing with **`end`**:

```
<%= form_with(model: @user, local: true) do |f| %>
  .
  .
  .
<% end %>
```

The presence of the **`do`** keyword indicates that **`form_with`** takes a block with one variable, which we've called **`f`** (for "form"). Note the presence of the hash argument **`local: true`**; by default, **`form_with`** sends a "remote" XHR request, whereas we want a regular "local" form request, mostly so that our error messages will render properly (Section 7.3.3).

As is usually the case with Rails helpers, we don't need to know any details about the implementation, but what we *do* need to know is what the **`f`** object does: when called with a method corresponding to an HTML form element—such as a text field, radio button, or password field—**`f`** returns code for that element specifically designed to set an attribute of the **`@user`** object. In other words,

```
<%= f.label :name %>
<%= f.text_field :name %>
```

creates the HTML needed to make a labeled text field element appropriate for setting the **`name`** attribute of a User model.

If you look at the HTML for the generated form by Ctrl-clicking and using the "inspect element" function of your browser, the page's source should look something like Listing 7.17. Let's take a moment to discuss its structure.

Listing 7.17: The HTML for the form in Figure 7.13.

```
<form accept-charset="UTF-8" action="/users" class="new_user"
      id="new_user" method="post">
  <input name="authenticity_token" type="hidden"
         value="NNb6+J/j46LcrgYUC60wQ2titMuJQ5lLqyAbnbAUkdo=" />
  <label for="user_name">Name</label>
  <input id="user_name" name="user[name]" type="text" />

  <label for="user_email">Email</label>
  <input id="user_email" name="user[email]" type="email" />

  <label for="user_password">Password</label>
  <input id="user_password" name="user[password]"
         type="password" />

  <label for="user_password_confirmation">Confirmation</label>
  <input id="user_password_confirmation"
         name="user[password_confirmation]" type="password" />

  <input class="btn btn-primary" name="commit" type="submit"
         value="Create my account" />
</form>
```

We'll start with the internal structure of the document. Comparing Listing 7.15 with Listing 7.17, we see that the embedded Ruby

```
<%= f.label :name %>
<%= f.text_field :name %>
```

produces the HTML

```
<label for="user_name">Name</label>
<input id="user_name" name="user[name]" type="text" />
```

while

```
<%= f.label :email %>
<%= f.email_field :email %>
```

produces the HTML

```
<label for="user_email">Email</label>
<input id="user_email" name="user[email]" type="email" />
```

and

```
<%= f.label :password %>
<%= f.password_field :password %>
```

produces the HTML

```
<label for="user_password">Password</label>
<input id="user_password" name="user[password]" type="password" />
```

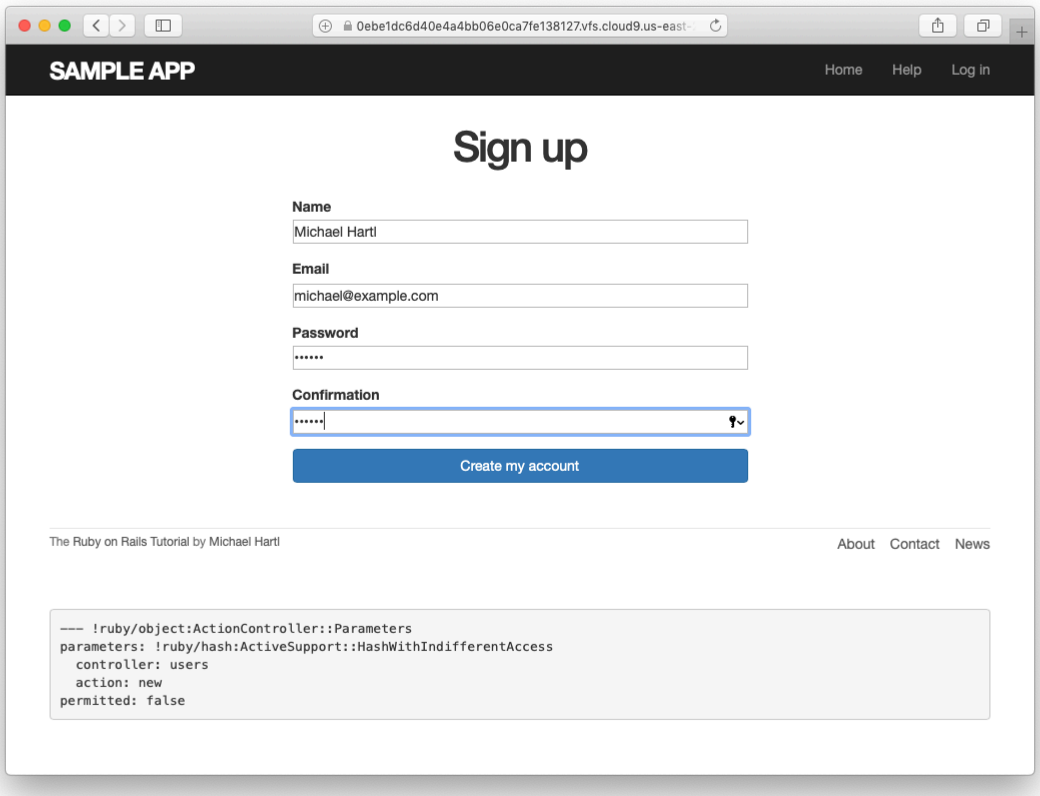
As seen in Figure 7.14, text and email fields (**`type="text"`** and **`type="email"`**) simply display their contents, whereas password fields (**`type="password"`**) obscure the input for security purposes, as seen in Figure 7.14. (The benefit of using an email field is that some systems treat it differently from a text field; for example, the code **`type="email"`** will cause some mobile devices to display a special keyboard optimized for entering email addresses.)

As we'll see in Section 7.4, the key to creating a user is the special **`name`** attribute in each **`input`**:

```
<input id="user_name" name="user[name]" - - - />
.
.
.
<input id="user_password" name="user[password]" - - - />
```

These **`name`** values allow Rails to construct an initialization hash (via the **`params`** variable) for creating users using the values entered by the user, as we'll see in Section 7.3.

The second important element is the **`form`** tag itself. Rails creates the **`form`** tag using the **`@user`** object: because every Ruby object knows its own class



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

Figure 7.14: A filled-in form with `text` and `password` fields.

(Section 4.4.1), Rails figures out that **`@user`** is of class **`User`**; moreover, since **`@user`** is a *new* user, Rails knows to construct a form with the **`post`** method, which is the proper verb for creating a new object (Box 3.2):

```
<form action="/users" class="new_user" id="new_user" method="post">
```

Here the **`class`** and **`id`** attributes are largely irrelevant; what's important is **`action="/users"`** and **`method="post"`**. Together, these constitute instructions to issue an HTTP `POST` request to the /users URL. We'll see in the next two sections what effects this has.

(You may also have noticed the code that appears just inside the **`form`** tag:

```
<input name="authenticity_token" type="hidden"
       value="NNb6+J/j46LcrgYUC60wQ2titMuJQ5lLqyAbnbAUkdo=" />
```

This code, which isn't displayed in the browser, is used internally by Rails, so it's not important for us to understand what it does. Briefly, it includes an *authenticity token*, which Rails uses to thwart an attack called a *cross-site request forgery* (CSRF). Knowing when it's OK to ignore details like this is a good mark of technical sophistication (Box 1.2).)[11]

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. *Learn Enough HTML to Be Dangerous*, in which all HTML is written by hand, doesn't cover the **`form`** tag. Why not?

[11]See the Stack Overflow entry on the Rails authenticity token if you're interested in the details of how this works.