

# Chapter 12

## Password reset

Having completed account activation (and thereby verified the user’s email address) in [Chapter 11](#), we’re now in a good position to implement *password reset*, and thereby handle the common case of users forgetting their passwords.<sup>1</sup> As we’ll see, many of the steps are similar to those for account activation, and we will have several opportunities to apply the lessons learned in [Chapter 11](#). The beginning is different, though; unlike account activation, implementing password resets requires both a change to one of our views and two new forms (to handle email and new password submission).

Before writing any code, let’s mock up the expected sequence for resetting passwords. We’ll start by adding a “forgot password” link to the sample application’s login form ([Figure 12.1](#)). The “forgot password” link will go to a page with a form that takes in an email address and sends an email containing a password reset link ([Figure 12.2](#)). The reset link will go to a form for resetting the user’s password (with confirmation), as shown in [Figure 12.3](#).

If you followed [Chapter 11](#), you already have a mailer for password resets, which was generated in [Section 11.2](#) ([Listing 11.6](#)). In this section, we’ll complete the necessary preliminaries by adding a resource and data model for password resets ([Section 12.1](#)) to go along with the mailer. We’ll implement the actual password reset in [Section 12.3](#).

In analogy with account activations, our general plan is to make a Pass-

---

<sup>1</sup>This chapter is independent of the others apart from using the mailer generation in [Listing 11.6](#), but it closely parallels [Chapter 11](#), so it’s much easier if you’ve followed that chapter first.

The image shows a login form within a rectangular border. At the top, a rounded horizontal bar contains three blue links: [Home](#), [Help](#), and [Log in](#). Below this bar, the text "Log in" is centered in a large, bold font. Underneath, the label "Email" is followed by a rectangular input field. Below that, the label "Password" is followed by a blue link [\(forgot password\)](#) and another rectangular input field. A checkbox is positioned to the left of the text "Remember me on this computer". Below the checkbox is a rounded button labeled "Log in". At the bottom of the form, the text "New user?" is followed by a blue link [Sign up now!](#). A second rounded horizontal bar is located at the very bottom of the form area.

Figure 12.1: A mockup of a “forgot password” link.

The image shows a wireframe of a web form titled "Forgot password". At the top, there is a horizontal navigation bar with three links: "Home", "Help", and "Log in". The main heading "Forgot password" is centered. Below it is a text input field labeled "Email". Underneath the input field is a "Submit" button. At the bottom of the form area, there is a long, empty rounded rectangular box.

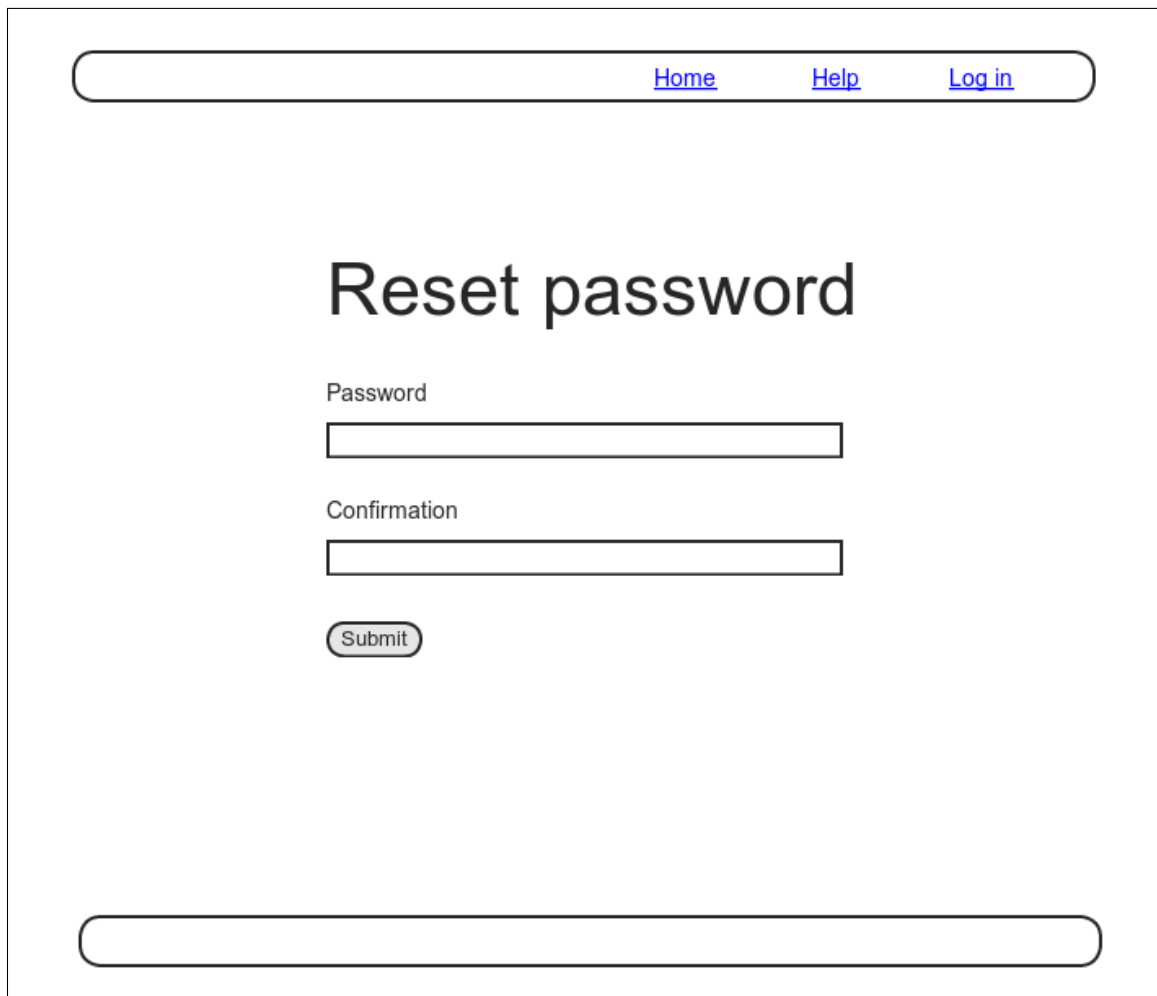
[Home](#)   [Help](#)   [Log in](#)

## Forgot password

Email

Submit

Figure 12.2: A mockup of the “forgot password” form.



The image shows a mockup of a password reset form. At the top, there is a horizontal navigation bar with three links: [Home](#), [Help](#), and [Log in](#). Below this bar, the main heading is "Reset password". Underneath the heading, there are two input fields: the first is labeled "Password" and the second is labeled "Confirmation". Below the "Confirmation" field is a "Submit" button. At the bottom of the form, there is a long, empty horizontal bar.

Figure 12.3: A mockup of the reset password form.

word Resets resource, with each password reset consisting of a reset token and corresponding reset digest. The primary sequence goes like this:

1. When a user requests a password reset, find the user by the submitted email address.
2. If the email address exists in the database, generate a reset token and corresponding reset digest.
3. Save the reset digest to the database, and then send an email to the user with a link containing the reset token and user's email address.
4. When the user clicks the link, find the user by email address, and then authenticate the token by comparing it to the reset digest.
5. If authenticated, present the user with the form for changing the password.

## 12.1 Password resets resource

As with sessions ([Section 8.1](#)) and account activations ([Chapter 11](#)), we'll model password resets as a resource even though they won't be associated with an Active Record model. Instead, we'll include the relevant data (including the reset token) in the User model itself.

Because we'll be treating password resets as a resource, we'll interact with them via the standard REST URLs. Unlike the activation link, which required only an **edit** action, in this case we'll be rendering both **new** and **edit** forms for manipulating password resets, as well as creating and updating them, so we'll end up using four RESTful routes in total.

As usual, we'll make a topic branch for the new feature:

```
$ git checkout -b password-reset
```

## 12.1.1 Password resets controller

Our first step is to generate a controller for the Password Resets resource, in this case making both **new** and **edit** actions per the discussion above:

```
$ rails generate controller PasswordResets new edit --no-test-framework
```

Note that we've included a flag to skip generating tests. This is because we don't need the controller tests, preferring instead to build on the integration test from [Section 11.3.3](#).

Because we'll need forms both for creating new password resets ([Figure 12.2](#)) and for updating them by changing the password in the User model ([Figure 12.3](#)), we need routes for **new**, **create**, **edit**, and **update**. We can arrange this with the **resources** line shown in [Listing 12.1](#).

### Listing 12.1: Adding a resource for password resets.

*config/routes.rb*

```
Rails.application.routes.draw do
  root 'static_pages#home'
  get  '/help',    to: 'static_pages#help'
  get  '/about',  to: 'static_pages#about'
  get  '/contact', to: 'static_pages#contact'
  get  '/signup', to: 'users#new'
  get  '/login',  to: 'sessions#new'
  post '/login',  to: 'sessions#create'
  delete '/logout', to: 'sessions#destroy'
  resources :users
  resources :account_activations, only: [:edit]
  resources :password_resets,    only: [:new, :create, :edit, :update]
end
```

The code in [Listing 12.1](#) arranges for the RESTful routes shown in [Table 12.1](#). In particular, the first route in [Table 12.1](#) gives a link to the “forgot password” form via

HTTP request	URL	Action	Named route
GET	/password_resets/new	<b>new</b>	<b>new_password_reset_path</b>
POST	/password_resets	<b>create</b>	<b>password_resets_path</b>
GET	/password_resets/<token>/edit	<b>edit</b>	<b>edit_password_reset_url(token)</b>
PATCH	/password_resets/<token>	<b>update</b>	<b>password_reset_path(token)</b>

Table 12.1: RESTful routes provided by the Password Resets resource in Listing 12.1.

```
new_password_reset_path
```

as seen in Listing 12.2 and Figure 12.4.

### Listing 12.2: Adding a link to password resets.

*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 %>
      <%= link_to "(forgot password)", new_password_reset_path %>
      <%= 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>
```

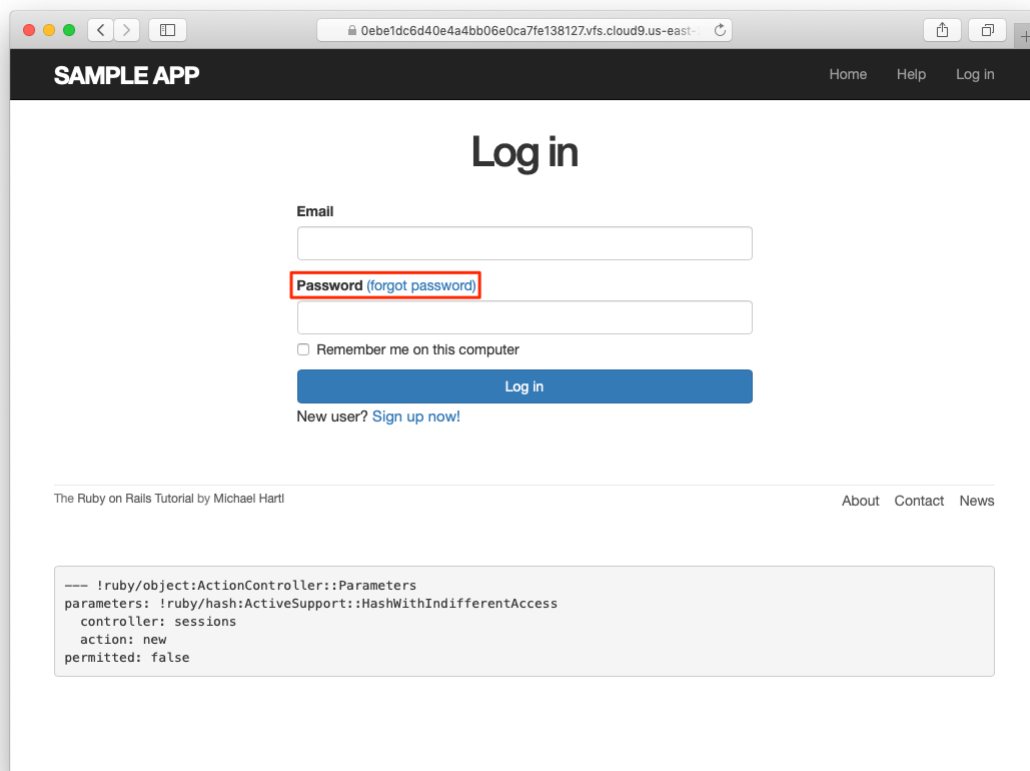


Figure 12.4: The login page with a “forgot password” link.



## 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. Verify that the test suite is still **GREEN**.
2. Why does [Table 12.1](#) list the **\_url** form of the **edit** named route instead of the **\_path** form? *Hint:* The answer is the same as for the similar account activations exercise ([Section 11.1.1](#)).

### 12.1.2 New password resets

To create new password resets, we first need to define the data model, which is similar to the one used for account activation ([Figure 11.1](#)). Following the pattern set by remember tokens ([Chapter 9](#)) and account activation tokens ([Chapter 11](#)), password resets will pair a virtual reset token for use in the reset email with a corresponding reset digest for retrieving the user. If we instead stored an unhashed token, an attacker with access to the database could send a reset request to the user's email address and then use the token and email to visit the corresponding password reset link, thereby gaining control of the account. Using a digest for password resets is thus essential. As an additional security precaution, we'll also plan to *expire* the reset link after a couple of hours, which requires recording the time when the reset gets sent. The resulting **reset\_digest** and **reset\_sent\_at** attributes appear in [Figure 12.5](#).

The migration to add the attributes from [Figure 12.5](#) appears as follows:

```
$ rails generate migration add_reset_to_users reset_digest:string \  
> reset_sent_at:datetime
```

(As in [Section 11.1.2](#), the **>** on the second line is a “line continuation” character inserted automatically by the shell, and should not be typed literally.) We then migrate as usual:

<b>users</b>	
<b>id</b>	integer
<b>name</b>	string
<b>email</b>	string
<b>created_at</b>	datetime
<b>updated_at</b>	datetime
<b>password_digest</b>	string
<b>remember_digest</b>	string
<b>admin</b>	boolean
<b>activation_digest</b>	string
<b>activated</b>	boolean
<b>activated_at</b>	datetime
<b>reset_digest</b>	string
<b>reset_sent_at</b>	datetime

Figure 12.5: The User model with added password reset attributes.

```
$ rails db:migrate
```

To make the view for new password resets, we'll work in analogy with the previous form for making a new non-Active Record resource, namely, the login form (Listing 8.4) for creating a new session, shown again in Listing 12.3 for reference.

**Listing 12.3:** Reviewing the code for 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 %>
      <%= link_to "(forgot password)", new_password_reset_path %>
      <%= 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>
```

The new password resets form has a lot in common with Listing 12.3; the most important differences are the use of a different resource and URL in the call to `form_with` and the omission of the password attribute. The result appears in Listing 12.4 and Figure 12.6.

**Listing 12.4:** A new password reset view.*app/views/password\_resets/new.html.erb*

```
<% provide(:title, "Forgot password") %>
<h1>Forgot password</h1>

<div class="row">
  <div class="col-md-6 col-md-offset-3">
    <%= form_with(url: password_resets_path, scope: :password_reset,
                 local: true) do |f| %>
      <%= f.label :email %>
      <%= f.email_field :email, class: 'form-control' %>

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

## 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. Why does the `form_with` in Listing 12.4 use `:password_reset` instead of `@password_reset`?

### 12.1.3 Password reset create action

Upon submitting the form in [Figure 12.6](#), we need to find the user by email address and update its attributes with the password reset token and sent-at timestamp. We then redirect to the root URL with an informative flash message. As with login ([Listing 8.11](#)), in the case of an invalid submission we re-render the `new` page with a `flash.now` message.<sup>2</sup> The results appear in [Listing 12.5](#).

---

<sup>2</sup>Security concerns about revealing the absence of the given email address are commonly misplaced. The reason is that this property is already present in every website that won't let you sign up with an email address that's already in use, which is practically all of them. Thus, indicating that a given email address isn't available for password resets gives potential attackers no additional information.

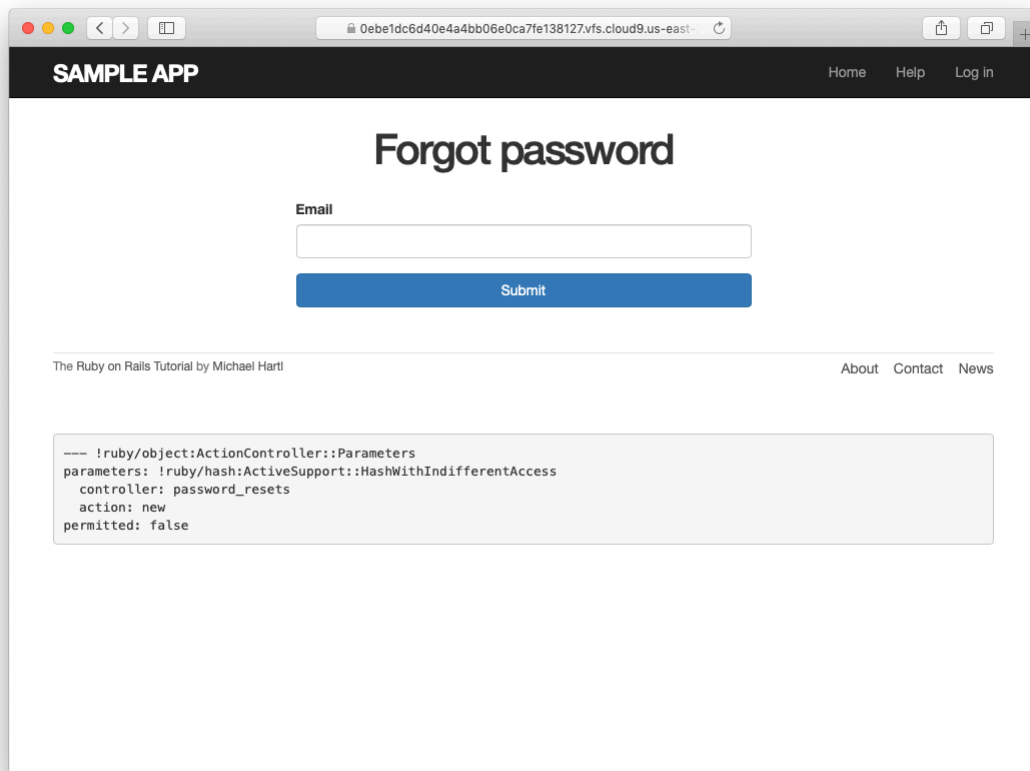


Figure 12.6: The “forgot password” form.

**Listing 12.5:** A **create** action for password resets.

*app/controllers/password\_resets\_controller.rb*

```
class PasswordResetsController < ApplicationController

  def new
  end

  def create
    @user = User.find_by(email: params[:password_reset][:email].downcase)
    if @user
      @user.create_reset_digest
      @user.send_password_reset_email
      flash[:info] = "Email sent with password reset instructions"
      redirect_to root_url
    else
      flash.now[:danger] = "Email address not found"
      render 'new'
    end
  end

  def edit
  end
end
```

The code in the User model parallels the **create\_activation\_digest** method used in the **before\_create** callback (Listing 11.3), as seen in Listing 12.6.

**Listing 12.6:** Adding password reset methods to the User model.

*app/models/user.rb*

```
class User < ApplicationRecord
  attr_accessor :remember_token, :activation_token, :reset_token
  before_save :downcase_email
  before_create :create_activation_digest
  .
  .
  .
  # Activates an account.
  def activate
    update_attribute(:activated, true)
    update_attribute(:activated_at, Time.zone.now)
  end
end

# Sends activation email.
```

```
def send_activation_email
  UserMailer.account_activation(self).deliver_now
end

# Sets the password reset attributes.
def create_reset_digest
  self.reset_token = User.new_token
  update_attribute(:reset_digest, User.digest(reset_token))
  update_attribute(:reset_sent_at, Time.zone.now)
end

# Sends password reset email.
def send_password_reset_email
  UserMailer.password_reset(self).deliver_now
end

private

# Converts email to all lower-case.
def downcase_email
  self.email = email.downcase
end

# Creates and assigns the activation token and digest.
def create_activation_digest
  self.activation_token = User.new_token
  self.activation_digest = User.digest(activation_token)
end
end
```

As shown in [Figure 12.7](#), at this point the application's behavior for invalid email addresses is already working. To get the application working upon submission of a valid email address as well, we need to define a password reset mailer method.

## 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. Submit a valid email address to the form shown in [Figure 12.6](#). What error message do you get?

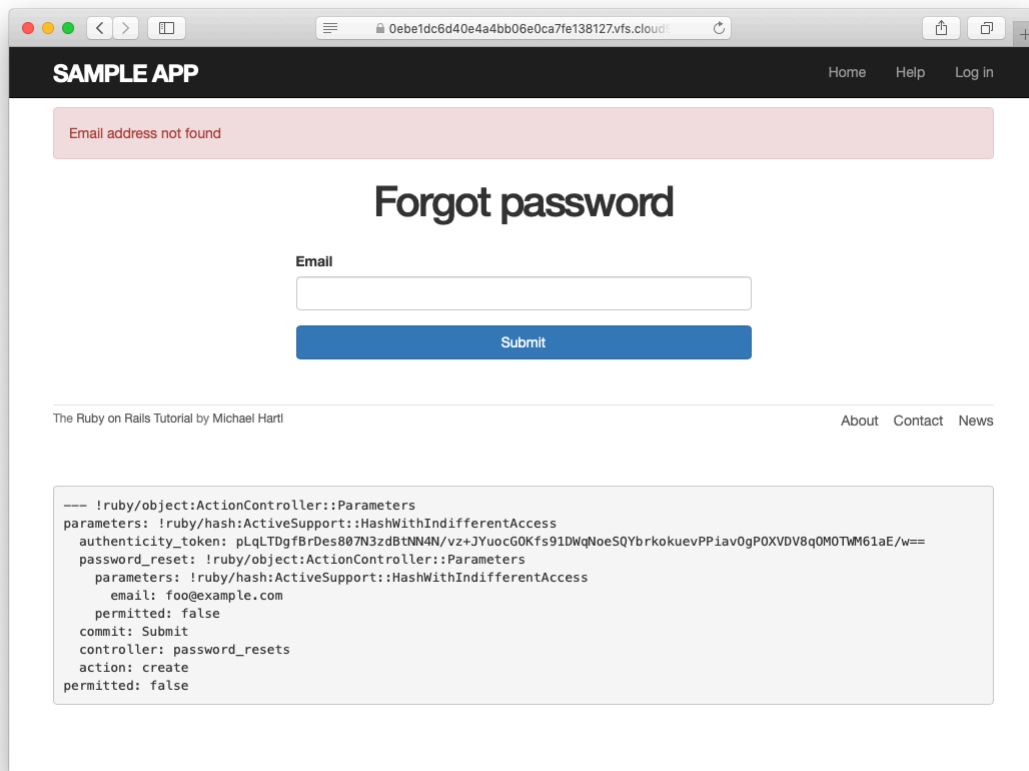


Figure 12.7: The “forgot password” form for an invalid email address.



2. Confirm at the console that the user in the previous exercise has valid `reset_digest` and `reset_sent_at` attributes, despite the error. What are the attribute values?

## 12.2 Password reset emails

We left [Section 12.1](#) with a nearly working `create` action in the Password Resets controller. The only thing missing is the method to deliver valid password reset emails.

If you followed [Section 11.1](#), you already have a default `password_reset` method in `app/mailers/user_mailer.rb` as a result of the User mailer generation in [Listing 11.6](#). If you skipped [Chapter 11](#), you can just copy the code below (omitting the `account_activation` and related methods) and create the missing files as necessary.

### 12.2.1 Password reset mailer and templates

In [Listing 12.6](#), we applied the design pattern implemented as a refactoring in [Section 11.3.3](#) by putting the User mailer directly in the model ([Listing 12.6](#)):

```
UserMailer.password_reset(self).deliver_now
```

The password reset mailer method needed to get this working is nearly identical to the mailer for account activation developed in [Section 11.2](#). We first create a `password_reset` method in the user mailer ([Listing 12.7](#)), and then define view templates for plain-text email ([Listing 12.8](#)) and HTML email ([Listing 12.9](#)).

**Listing 12.7:** Mailing the password reset link.

```
app/mailers/user_mailer.rb
```

```
class UserMailer < ApplicationMailer
```

```
def account_activation(user)
  @user = user
  mail to: user.email, subject: "Account activation"
end

def password_reset(user)
  @user = user
  mail to: user.email, subject: "Password reset"
end
end
```

**Listing 12.8:** The password reset plain-text email template.

*app/views/user\_mailer/password\_reset.text.erb*

To reset your password click the link below:

```
<%= edit_password_reset_url(@user.reset_token, email: @user.email) %>
```

This link will expire in two hours.

If you did not request your password to be reset, please ignore this email and your password will stay as it is.

**Listing 12.9:** The password reset HTML email template.

*app/views/user\_mailer/password\_reset.html.erb*

```
<h1>Password reset</h1>
```

```
<p>To reset your password click the link below:</p>
```

```
<%= link_to "Reset password", edit_password_reset_url(@user.reset_token,
  email: @user.email) %>
```

```
<p>This link will expire in two hours.</p>
```

```
<p>
```

```
If you did not request your password to be reset, please ignore this email and
your password will stay as it is.
```

```
</p>
```

As with account activation emails (Section 11.2), we can preview password reset emails using the Rails email previewer. The code is exactly analogous to Listing 11.18, as shown in Listing 12.10.

**Listing 12.10:** A working preview method for password reset.

*test/mailers/previews/user\_mailer\_preview.rb*

```
# Preview all emails at http://localhost:3000/rails/mailers/user_mailer
class UserMailerPreview < ActionMailer::Preview

  # Preview this email at
  # http://localhost:3000/rails/mailers/user_mailer/account_activation
  def account_activation
    user = User.first
    user.activation_token = User.new_token
    UserMailer.account_activation(user)
  end

  # Preview this email at
  # http://localhost:3000/rails/mailers/user_mailer/password_reset
  def password_reset
    user = User.first
    user.reset_token = User.new_token
    UserMailer.password_reset(user)
  end
end
```

With the code in Listing 12.10, the HTML and text email previews appear as in Figure 12.8 and Figure 12.9.

With the code in Listing 12.7, Listing 12.8, and Listing 12.9, submission of a valid email address appears as shown in Figure 12.10. The corresponding email appears in the server log and should look something like Listing 12.11.

**Listing 12.11:** A sample password reset email from the server log.

```
UserMailer#password_reset: processed outbound mail in 6.0ms
Delivered mail 5d609328d5d29_28872b106582ddf4886d8@ip-172-31-25-202.mail (2.8ms)
Date: Sat, 24 Aug 2019 01:30:16 +0000
From: noreply@example.com
To: michael@michaelhartl.com
Message-ID: <5d609328d5d29_28872b106582ddf4886d8@ip-172-31-25-202.mail>
Subject: Password reset
Mime-Version: 1.0
Content-Type: multipart/alternative;
  boundary="---=_mimepart_5d609328d5404_28872b106582ddf488531";
  charset=UTF-8
Content-Transfer-Encoding: 7bit
```

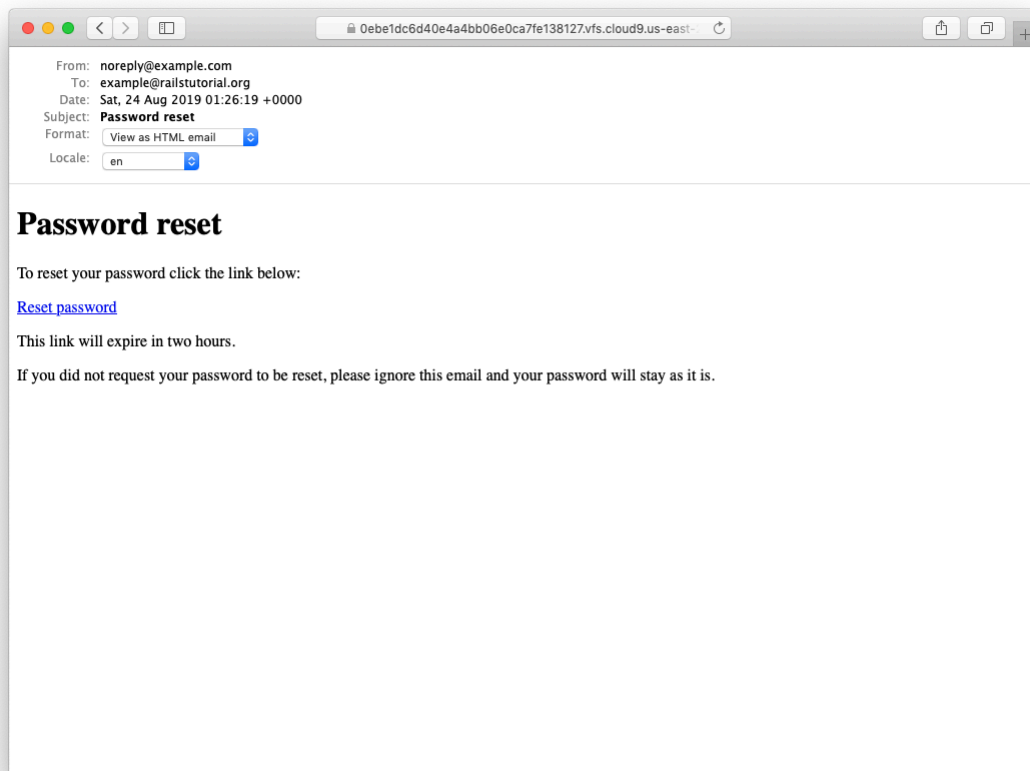


Figure 12.8: A preview of the HTML version of the password reset email.

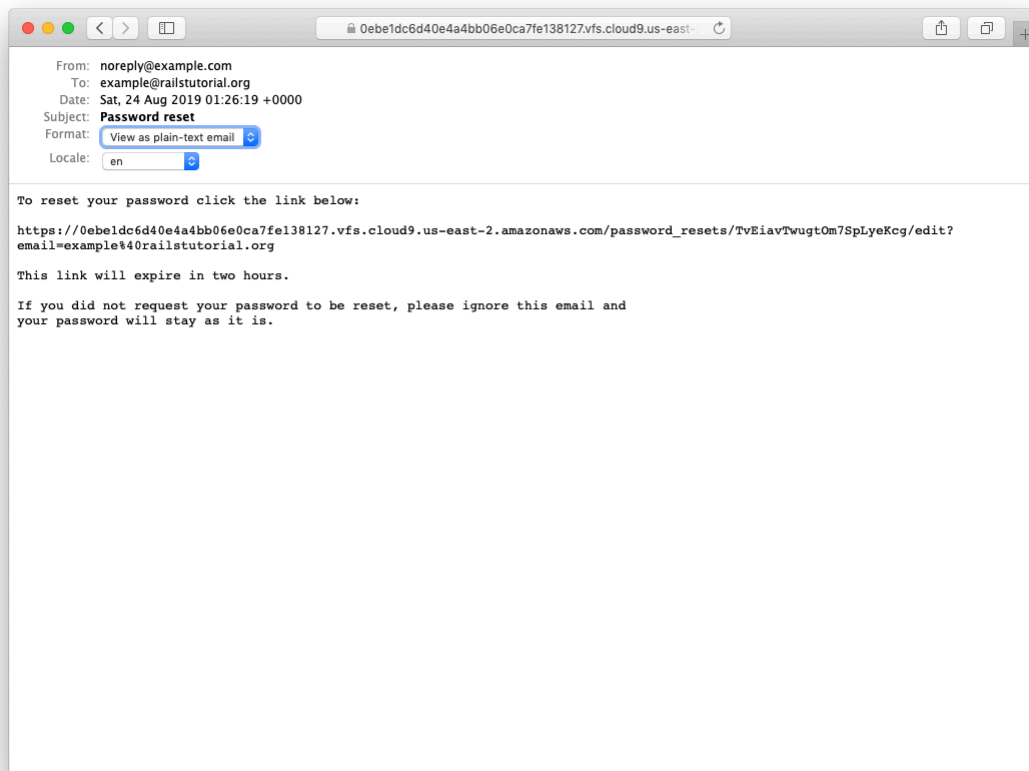


Figure 12.9: A preview of the text version of the password reset email.

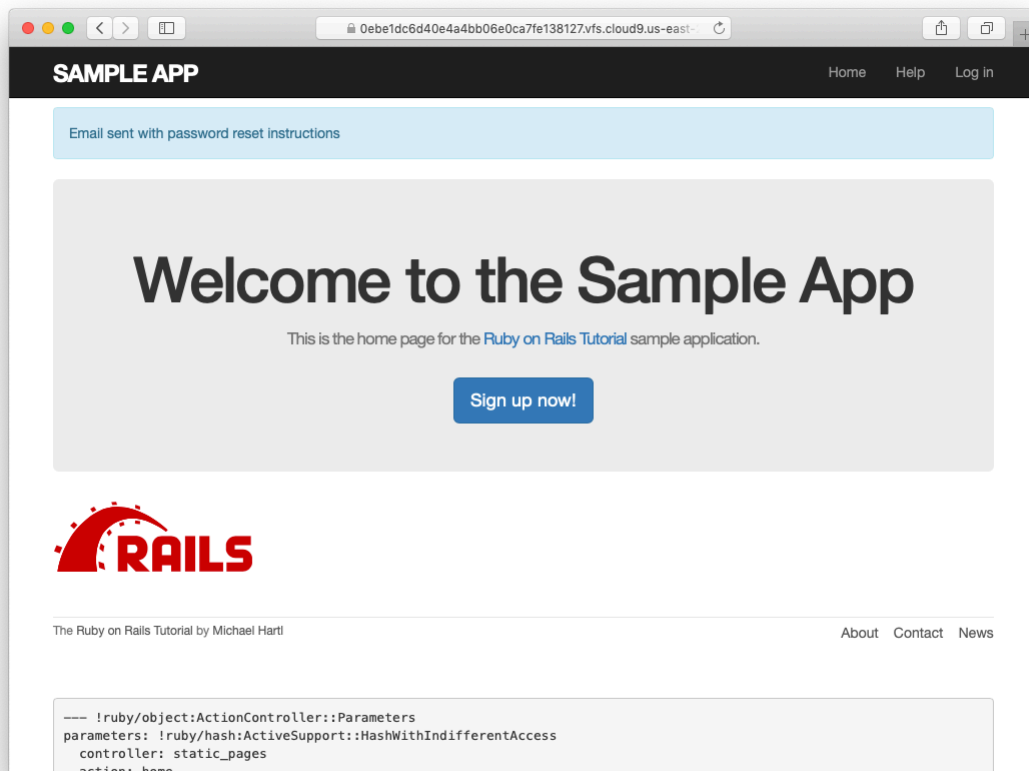


Figure 12.10: The result of submitting a valid email address.

```
-----=_mimepart_5d609328d5404_28872b106582ddf488531
Content-Type: text/plain;
  charset=UTF-8
Content-Transfer-Encoding: 7bit

To reset your password click the link below:

https://0ebeldc6d40e4a4bb06e0ca7fe138127.vfs.cloud9.us-east-2.
amazonaws.com/password_resets/cT3mB4pwu7o-hrg6qEDfKg/
edit?email=michael%40michaelhartl.com

This link will expire in two hours.

If you did not request your password to be reset, please ignore this email and
your password will stay as it is.

-----=_mimepart_5d609328d5404_28872b106582ddf488531
Content-Type: text/html;
  charset=UTF-8
Content-Transfer-Encoding: 7bit

<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style>
      /* Email styles need to be inline */
    </style>
  </head>

  <body>
    <h1>Password reset</h1>

<p>To reset your password click the link below:</p>

<a href="https://0ebeldc6d40e4a4bb06e0ca7fe138127.vfs.cloud9.us-east-2.
amazonaws.com/password_resets/cT3mB4pwu7o-hrg6qEDfKg/
edit?email=michael%40michaelhartl.com">Reset password</a>

<p>This link will expire in two hours.</p>

<p>
If you did not request your password to be reset, please ignore this email and
your password will stay as it is.
</p>
  </body>
</html>

-----=_mimepart_5d609328d5404_28872b106582ddf488531--
```

## 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. Preview the email templates in your browser. What do the Date fields read for your previews?
2. Submit a valid email address to the new password reset form. What is the content of the generated email in the server log?
3. At the console, find the user object corresponding to the email address from the previous exercise and verify that it has valid `reset_digest` and `reset_sent_at` attributes.

### 12.2.2 Email tests

In analogy with the account activation mailer method test ([Listing 11.20](#)), we'll write a test of the password reset mailer method, as shown in [Listing 12.12](#).

**Listing 12.12:** Adding a test of the password reset mailer method. GREEN

*test/mailers/user\_mailer\_test.rb*

```
require 'test_helper'

class UserMailerTest < ActionMailer::TestCase

  test "account_activation" do
    user = users(:michael)
    user.activation_token = User.new_token
    mail = UserMailer.account_activation(user)
    assert_equal "Account activation", mail.subject
    assert_equal [user.email], mail.to
    assert_equal ["noreply@example.com"], mail.from
    assert_match user.name, mail.body.encoded
    assert_match user.activation_token, mail.body.encoded
    assert_match CGI.escape(user.email), mail.body.encoded
  end

  test "password_reset" do
```



```
user = users(:michael)
user.reset_token = User.new_token
mail = UserMailer.password_reset(user)
assert_equal "Password reset", mail.subject
assert_equal [user.email], mail.to
assert_equal ["noreply@example.com"], mail.from
assert_match user.reset_token, mail.body.encoded
assert_match CGI.escape(user.email), mail.body.encoded
end
end
```

At this point, the test suite should be **GREEN**:

#### Listing 12.13: **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. Run just the mailer tests. Are they **GREEN**?
2. Confirm that the test goes **RED** if you remove the second call to **CGI.escape** in [Listing 12.12](#).

## 12.3 Resetting the password

Now that we have a correctly generated email as in [Listing 12.11](#), we need to write the **edit** action in the Password Resets controller that actually resets the user's password. As in [Section 11.3.3](#), we'll write a thorough integration test as well.

### 12.3.1 Reset `edit` action

Password reset emails such as that shown in [Listing 12.11](#) contain links of the following form:

```
https://example.com/password_resets/3BdBrXeQZSWqFIDRN8cxHA/edit?email=fu%40bar.com
```

To get these links to work, we need a form for resetting passwords. The task is similar to updating users via the user edit view ([Listing 10.2](#)), but in this case with only password and confirmation fields.

There's an additional complication, though: we expect to find the user by email address, which means we need its value in both the `edit` and `update` actions. The email will automatically be available in the `edit` action because of its presence in the link above, but after we submit the form its value will be lost. The solution is to use a *hidden field* to place (but not display) the email on the page, and then submit it along with the rest of the form's information. The result appears in [Listing 12.14](#).

#### Listing 12.14: The form to reset a password.

```
app/views/password_resets/edit.html.erb
```

```
<%= provide(:title, 'Reset password') %>
<h1>Reset password</h1>

<div class="row">
  <div class="col-md-6 col-md-offset-3">
    <%= form_with(model: @user, url: password_reset_path(params[:id]),
                  local: true) do |f| %>
      <%= render 'shared/error_messages' %>

      <%= hidden_field_tag :email, @user.email %>

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

      <%= f.label :password_confirmation, "Confirmation" %>
      <%= f.password_field :password_confirmation, class: 'form-control' %>

      <%= f.submit "Update password", class: "btn btn-primary" %>
    <% end %>
  </div>
</div>
```

Note that [Listing 12.14](#) uses the form tag helper

```
hidden_field_tag :email, @user.email
```

instead of

```
f.hidden_field :email, @user.email
```

because the reset link puts the email in `params[:email]`, whereas the latter would put it in `params[:user][:email]`.

To get the form to render, we need to define an `@user` variable in the Password Resets controller's `edit` action. As with account activation ([Listing 11.31](#)), this involves finding the user corresponding to the email address in `params[:email]`. We then need to verify that the user is valid, i.e., that it exists, is activated, and is authenticated according to the reset token from `params[:id]` (using the generalized `authenticated?` method defined in [Listing 11.26](#)). Because the existence of a valid `@user` is needed in both the `edit` and `update` actions, we'll put the code to find and validate it in a couple of before filters, as shown in [Listing 12.15](#).

**Listing 12.15:** The `edit` action for password reset.

*app/controllers/password\_resets\_controller.rb*

```
class PasswordResetsController < ApplicationController
  before_action :get_user, only: [:edit, :update]
  before_action :valid_user, only: [:edit, :update]
  .
  .
  .
  def edit
  end

  private

  def get_user
    @user = User.find_by(email: params[:email])
  end
end
```

```

# Confirms a valid user.
def valid_user
  unless (@user && @user.activated? &&
    @user.authenticated?(:reset, params[:id]))
    redirect_to root_url
  end
end
end
end

```

In Listing 12.15, compare the use of

```
authenticated?(:reset, params[:id])
```

to

```
authenticated?(:remember, cookies[:remember_token])
```

in Listing 11.28 and

```
authenticated?(:activation, params[:id])
```

in Listing 11.31. Together, these three uses complete the authentication methods shown in Table 11.1.

With the code as above, following the link from Listing 12.11 should render a password reset form. The result of pasting the link from the log (Listing 12.11) appears in Figure 12.11.

## 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. Follow the link in the email from the server log in Section 12.2.1. Does it properly render the form as shown in Figure 12.11?
2. What happens if you submit the form from the previous exercise?

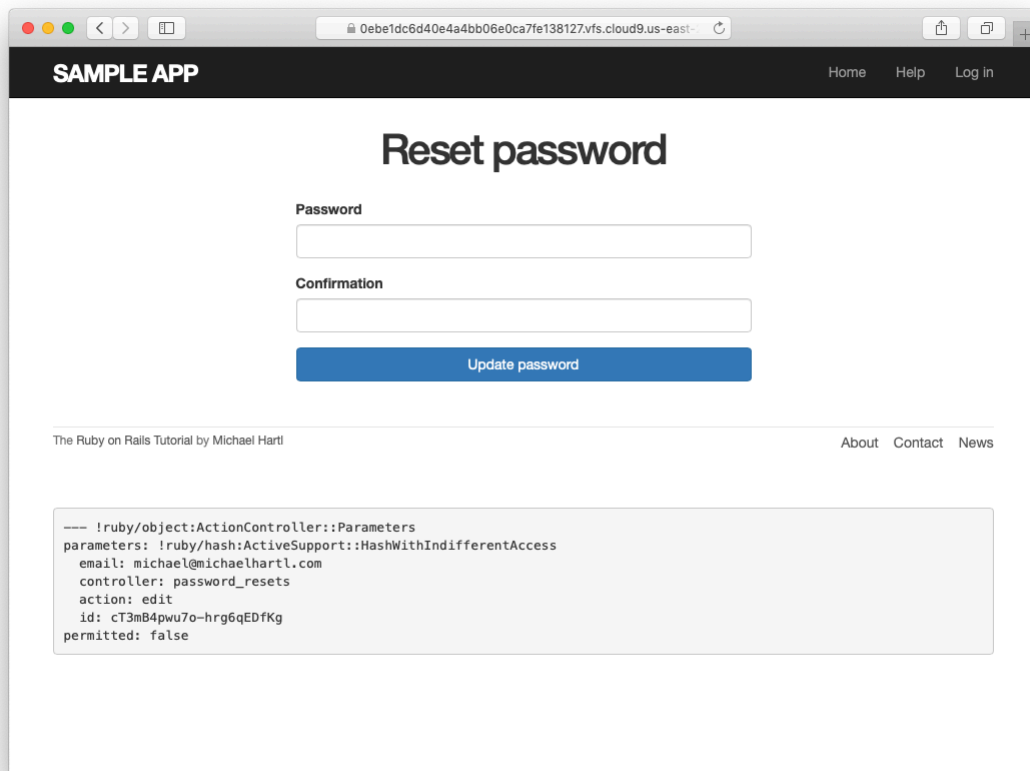


Figure 12.11: The password reset form.

## 12.3.2 Updating the reset

Unlike the Account Activations `edit` method, which simply toggles the user from “inactive” to “active”, the `edit` method for Password Resets is a form, which must therefore submit to a corresponding `update` action. To define this `update` action, we need to consider four cases:

1. An expired password reset
2. A failed update due to an invalid password
3. A failed update (which initially looks “successful”) due to an empty password and confirmation
4. A successful update

Cases (1), (2), and (4) are fairly straightforward, but Case (3) is non-obvious and is explained in more detail below.

Case (1) applies to both the `edit` and `update` actions, and so logically belongs in a before filter:

```
before_action :check_expiration, only: [:edit, :update] # Case (1)
```

This requires defining a private `check_expiration` method:

```
# Checks expiration of reset token.
def check_expiration
  if @user.password_reset_expired?
    flash[:danger] = "Password reset has expired."
    redirect_to new_password_reset_url
  end
end
```

In the `check_expiration` method, we’ve deferred the expiration check to the instance method `password_reset_expired?`, which is a little tricky and will be defined in a moment.

Listing 12.16 shows the implementation of these filters, together with the `update` action that implements Cases (2)–(4). Case (2) gets handled by a failed

update, with the error messages from the shared partial in [Listing 12.14](#) displaying automatically when the `edit` form is re-rendered. Case (4) corresponds to a successful change, and the result is similar to a successful login ([Listing 8.29](#)).

The only failure case not handled by Case (2) is when the password is empty, which is currently allowed by our User model ([Listing 10.13](#)) and so needs to be caught and handled explicitly.<sup>3</sup> This is Case (3) above. Our method in this case is to add an error directly to the `@user` object's error messages using `errors.add`:

```
@user.errors.add(:password, :blank)
```

This arranges to use the default message for blank content when the password is empty.<sup>4</sup>

The result of putting Cases (1)–(4) together is the `update` action shown in [Listing 12.16](#).

**Listing 12.16:** The `update` action for password reset.

*app/controllers/password\_resets\_controller.rb*

```
class PasswordResetsController < ApplicationController
  before_action :get_user,          only: [:edit, :update]
  before_action :valid_user,       only: [:edit, :update]
  before_action :check_expiration, only: [:edit, :update] # Case (1)

  def new
  end

  def create
    @user = User.find_by(email: params[:password_reset][:email].downcase)
    if @user
      @user.create_reset_digest
      @user.send_password_reset_email
      flash[:info] = "Email sent with password reset instructions"
      redirect_to root_url
    end
  end
end
```

<sup>3</sup>We need only handle the case where the password is empty because if the confirmation is empty, the confirmation validation (which is skipped if the password is empty) will catch the problem and supply a relevant error message.

<sup>4</sup>Alert reader Khaled Teilab has noted that one advantage of using `errors.add(:password, :blank)` is that the resulting message is automatically rendered in the correct language when using the `rails-i18n` gem.

```
    else
      flash.now[:danger] = "Email address not found"
      render 'new'
    end
  end

  def edit
  end

  def update
    if params[:user][:password].empty? # Case (3)
      @user.errors.add(:password, "can't be empty")
      render 'edit'
    elsif @user.update(user_params) # Case (4)
      log_in @user
      flash[:success] = "Password has been reset."
      redirect_to @user
    else # Case (2)
      render 'edit'
    end
  end

  private

  def user_params
    params.require(:user).permit(:password, :password_confirmation)
  end

  # Before filters

  def get_user
    @user = User.find_by(email: params[:email])
  end

  # Confirms a valid user.
  def valid_user
    unless (@user && @user.activated? &&
      @user.authenticated?(:reset, params[:id]))
      redirect_to root_url
    end
  end

  # Checks expiration of reset token.
  def check_expiration
    if @user.password_reset_expired?
      flash[:danger] = "Password reset has expired."
      redirect_to new_password_reset_url
    end
  end
end
```



Note that we've added a `user_params` method permitting both the password and password confirmation attributes (Section 7.3.2).

As noted above, the implementation in Listing 12.16 delegates the boolean test for password reset expiration to the User model via the code

```
@user.password_reset_expired?
```

To get this to work, we need to define the `password_reset_expired?` method. As indicated in the email templates from Section 12.2.1, we'll consider a password reset to be expired if it was sent more than two hours ago, which we can express in Ruby as follows:

```
reset_sent_at < 2.hours.ago
```

This can be confusing if you read `<` as “less than”, because then it sounds like “Password reset sent less than two hours ago”, which is the opposite of what we want. In this context, it's better to read `<` as “earlier than”, which gives something like “Password reset sent earlier than two hours ago.” That *is* what we want, and it leads to the `password_reset_expired?` method in Listing 12.17. (For a formal demonstration that the comparison is correct, see the proof in Section 12.6.)

**Listing 12.17:** Adding password reset methods to the User model.

*app/models/user.rb*

```
class User < ApplicationRecord
  .
  .
  .
  # Returns true if a password reset has expired.
  def password_reset_expired?
    reset_sent_at < 2.hours.ago
  end

  private
  .
  .
  .
end
```

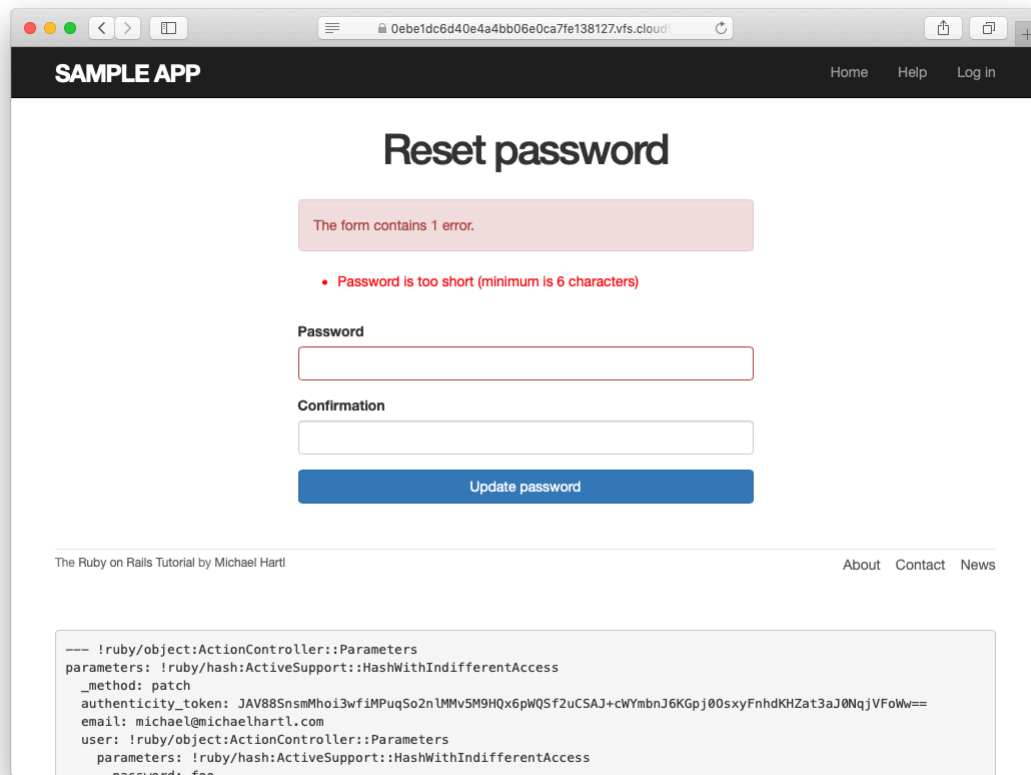


Figure 12.12: A failed password reset.

With the code in [Listing 12.17](#), the **update** action in [Listing 12.16](#) should be working. The results for invalid and valid submissions are shown in [Figure 12.12](#) and [Figure 12.13](#), respectively. (Lacking the patience to wait two hours, we’ll cover the third branch in a test, which is left as an exercise ([Section 12.3.3](#))).

## 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](#).

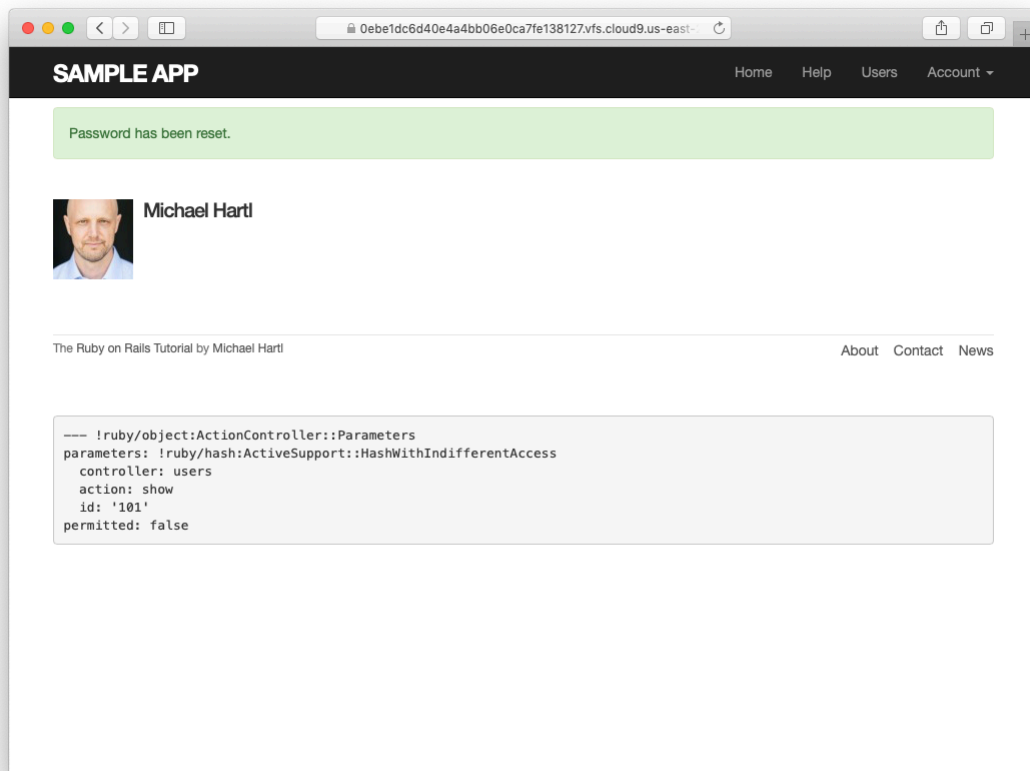


Figure 12.13: A successful password reset.

1. Follow the email link from [Section 12.2.1](#) again and submit mismatched passwords to the form. What is the error message?
2. In the console, find the user belonging to the email link, and retrieve the value of the `password_digest` attribute. Now submit valid matching passwords to the form shown in [Figure 12.12](#). Did the submission appear to work? How did it affect the value of `password_digest`? *Hint*: Use `user.reload` to retrieve the new value.

### 12.3.3 Password reset test

In this section, we'll write an integration test covering two of the three branches in [Listing 12.16](#), invalid and valid submission. (As noted above, testing the third branch is left as an exercise ([Section 12.3.3](#).) We'll get started by generating a test file for password resets:

```
$ rails generate integration_test password_resets
  invoke  test_unit
  create  test/integration/password_resets_test.rb
```

The steps to test password resets broadly parallel the test for account activation from [Listing 11.33](#), though there is a difference at the outset: we first visit the “forgot password” form and submit invalid and then valid email addresses, the latter of which creates a password reset token and sends the reset email. We then visit the link from the email and again submit invalid and valid information, verifying the correct behavior in each case. The resulting test, shown in [Listing 12.18](#), is an excellent exercise in reading code.

**Listing 12.18:** An integration test for password resets.

```
test/integration/password_resets_test.rb

require 'test_helper'

class PasswordResetsTest < ActionDispatch::IntegrationTest
  def setup
```

```
ActionMailer::Base.deliveries.clear
@user = users(:michael)
end

test "password resets" do
  get new_password_reset_path
  assert_template 'password_resets/new'
  assert_select 'input[name=?]', 'password_reset[email]'
  # Invalid email
  post password_resets_path, params: { password_reset: { email: "" } }
  assert_not flash.empty?
  assert_template 'password_resets/new'
  # Valid email
  post password_resets_path,
    params: { password_reset: { email: @user.email } }
  assert_not_equal @user.reset_digest, @user.reload.reset_digest
  assert_equal 1, ActionMailer::Base.deliveries.size
  assert_not flash.empty?
  assert_redirected_to root_url
  # Password reset form
  user = assigns(:user)
  # Wrong email
  get edit_password_reset_path(user.reset_token, email: "")
  assert_redirected_to root_url
  # Inactive user
  user.toggle!(:activated)
  get edit_password_reset_path(user.reset_token, email: user.email)
  assert_redirected_to root_url
  user.toggle!(:activated)
  # Right email, wrong token
  get edit_password_reset_path('wrong token', email: user.email)
  assert_redirected_to root_url
  # Right email, right token
  get edit_password_reset_path(user.reset_token, email: user.email)
  assert_template 'password_resets/edit'
  assert_select "input[name=email][type=hidden][value=?]", user.email
  # Invalid password & confirmation
  patch password_reset_path(user.reset_token),
    params: { email: user.email,
              user: { password: "foobaz",
                     password_confirmation: "barquux" } }
  assert_select 'div#error_explanation'
  # Empty password
  patch password_reset_path(user.reset_token),
    params: { email: user.email,
              user: { password: "",
                     password_confirmation: "" } }
  assert_select 'div#error_explanation'
  # Valid password & confirmation
  patch password_reset_path(user.reset_token),
    params: { email: user.email,
```

```

        user: { password: "foobaz",
                password_confirmation: "foobaz" } }
    assert is_logged_in?
    assert_not flash.empty?
    assert_redirected_to user
  end
end

```

Most of the ideas in [Listing 12.18](#) have appeared previously in this tutorial; the only really novel element is the test of the **input** tag:

```
assert_select "input[name=email][type=hidden][value=?]", user.email
```

This makes sure that there is an **input** tag with the right name, (hidden) type, and email address:

```
<input id="email" name="email" type="hidden" value="michael@example.com" />
```

With the code as in [Listing 12.18](#), our test suite should be **GREEN**:

### Listing 12.19: 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. In [Listing 12.6](#), the **create\_reset\_digest** method makes two calls to **update\_attribute**, each of which requires a separate database operation. By filling in the template shown in [Listing 12.20](#), replace the two **update\_attribute** calls with a single call to **update\_columns**,

which hits the database only once. After making the changes, verify that the test suite is still **GREEN**. (For convenience, [Listing 12.20](#) includes the results of solving the exercise in [Listing 11.39](#).)

2. Write an integration test for the expired password reset branch in [Listing 12.16](#) by filling in the template shown in [Listing 12.21](#). (This code introduces **response.body**, which returns the full HTML body of the page.) There are many ways to test for the result of an expiration, but the method suggested by [Listing 12.21](#) is to (case-insensitively) check that the response body includes the word “expired”.
3. Expiring password resets after a couple of hours is a nice security precaution, but there is an even more secure solution for cases where a public computer is used. The reason is that the password reset link remains active for 2 hours and can be used even if logged out. If a user reset their password from a public machine, anyone could press the back button and change the password (and get logged in to the site). To fix this, add the code shown in [Listing 12.22](#) to clear the reset digest on successful password update.<sup>5</sup>
4. Add a line to [Listing 12.18](#) to test for the clearing of the reset digest in the previous exercise. *Hint:* Combine **assert\_nil** (first seen in [Listing 9.25](#)) with **user.reload** ([Listing 11.33](#)) to test the **reset\_digest** attribute directly.

**Listing 12.20:** A template for using **update\_columns**.

*app/models/user.rb*

```
class User < ApplicationRecord
  attr_accessor :remember_token, :activation_token, :reset_token
  before_save :downcase_email
  before_create :create_activation_digest
  .
  .
  .
```

<sup>5</sup>Thanks to reader Tristan Ludowyk for suggesting this feature and for providing both a detailed description and a suggested implementation.

```
# Activates an account.
def activate
  update_columns(activated: true, activated_at: Time.zone.now)
end

# Sends activation email.
def send_activation_email
  UserMailer.account_activation(self).deliver_now
end

# Sets the password reset attributes.
def create_reset_digest
  self.reset_token = User.new_token
  update_columns(reset_digest: FILL_IN, reset_sent_at: FILL_IN)
end

# Sends password reset email.
def send_password_reset_email
  UserMailer.password_reset(self).deliver_now
end

private

# Converts email to all lower-case.
def downcase_email
  self.email = email.downcase
end

# Creates and assigns the activation token and digest.
def create_activation_digest
  self.activation_token = User.new_token
  self.activation_digest = User.digest(activation_token)
end
end
```

**Listing 12.21:** A test for an expired password reset. GREEN

*test/integration/password\_resets\_test.rb*

```
require 'test_helper'

class PasswordResetsTest < ActionDispatch::IntegrationTest

  def setup
    ActionMailer::Base.deliveries.clear
    @user = users(:michael)
  end

  .
  .
  .
```



```
test "expired token" do
  get new_password_reset_path
  post password_resets_path,
      params: { password_reset: { email: @user.email } }

  @user = assigns(:user)
  @user.update_attribute(:reset_sent_at, 3.hours.ago)
  patch password_reset_path(@user.reset_token),
      params: { email: @user.email,
              user: { password: "foobar",
                    password_confirmation: "foobar" } }

  assert_response :redirect
  follow_redirect!
  assert_match /FILL_IN/i, response.body
end
end
```

**Listing 12.22:** Clearing the reset digest on successful password reset.  
*app/controllers/password\_resets\_controller.rb*

```
class PasswordResetsController < ApplicationController
  .
  .
  .
  def update
    if params[:user][:password].empty?
      @user.errors.add(:password, "can't be empty")
      render 'edit'
    elsif @user.update(user_params)
      log_in @user
      @user.update_attribute(:reset_digest, nil)
      flash[:success] = "Password has been reset."
      redirect_to @user
    else
      render 'edit'
    end
  end
end
.
.
.
end
```

## 12.4 Email in production (take two)

Now that we've got password resets working in development, in this section we'll get them working in production as well. The steps are exactly the same as for account activations, so if you already followed [Section 11.4](#) you can skip right to [Listing 12.24](#).

To send email in production, we'll use SendGrid, which is available as an add-on at Heroku for verified accounts. (This requires adding credit card information to your Heroku account, but there is no charge when verifying an account.) For our purposes, the “starter” tier (which as of this writing is limited to 400 emails a day but costs nothing) is the best fit. We can add it to our app as follows:

```
$ heroku addons:create sendgrid:starter
```

(This might fail on systems with older version of Heroku's command-line interface. In this case, either [upgrade to the latest Heroku toolbelt](#) or try the older syntax **heroku addons:add sendgrid:starter**.)

To configure our application to use SendGrid, we need to fill out the [SMTP](#) settings for our production environment. As shown in [Listing 12.23](#), you will also have to define a **host** variable with the address of your production website.

### Listing 12.23: Configuring Rails to use SendGrid in production.

*config/environments/production.rb*

```
Rails.application.configure do
  .
  .
  .
  config.action_mailer.raise_delivery_errors = true
  config.action_mailer.delivery_method = :smtp
  host = '<your heroku app>.herokuapp.com'
  config.action_mailer.default_url_options = { host: host }
  ActionMailer::Base.smtp_settings = {
    :address      => 'smtp.sendgrid.net',
    :port         => '587',
    :authentication => :plain,
    :user_name    => ENV['SENDGRID_USERNAME'],
```

```
:password => ENV[ 'SENDGRID_PASSWORD' ],
:domain   => 'heroku.com',
:enable_starttls_auto => true
}
.
.
.
end
```

The email configuration in [Listing 11.41](#) includes the `user_name` and `password` of the SendGrid account, but note that they are accessed via the `ENV` environment variable instead of being hard-coded. This is a best practice for production applications, which for security reasons should never expose sensitive information such as raw passwords in source code. In the present case, these variables are configured automatically via the SendGrid add-on, but we'll see an example in [Section 13.4.4](#) where we'll have to define them ourselves.

At this point, you should merge the topic branch into master ([Listing 12.24](#)).

**Listing 12.24:** Merging the `password-reset` branch into `master`.

```
$ rails test
$ git add -A
$ git commit -m "Add password reset"
$ git checkout master
$ git merge password-reset
```

Then push up to the remote repository and deploy to Heroku:

```
$ rails test
$ git push && git push heroku
$ heroku run rails db:migrate
```

Once the Heroku deploy has finished, you can reset your password by clicking the “(forgot password)” link ([Figure 12.4](#)). The result should be a reset email as shown in [Figure 12.14](#). Following the link and making invalid or valid submissions should work as it did in development ([Figure 12.12](#) and [Figure 12.13](#)). Likewise, upon successfully changing the password, the user should be redirected to the profile page ([Figure 12.15](#)).

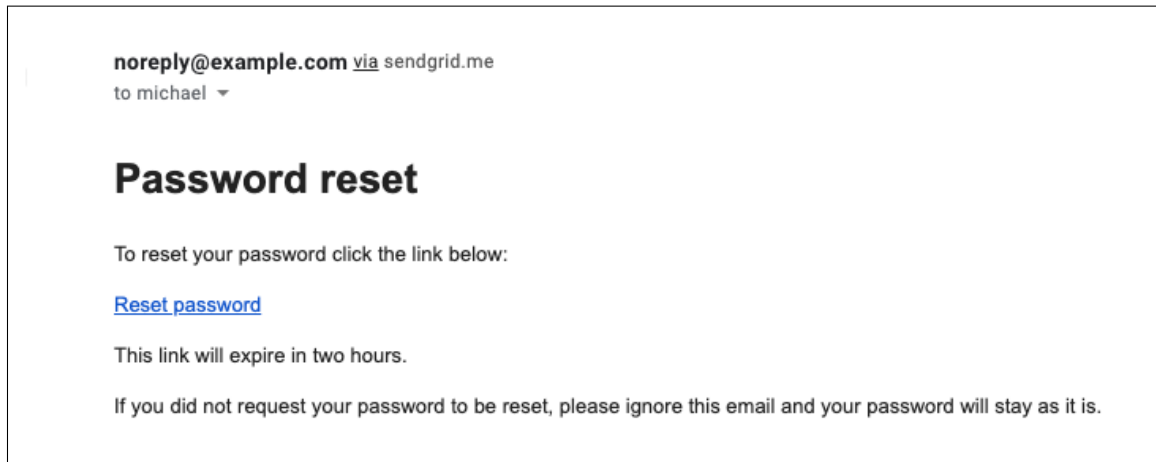


Figure 12.14: A password reset email sent in production.

## 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. Sign up for a new account in production. Did you get the email?
2. Click on the link in the activation email to confirm that it works. What is the corresponding entry in the server log? *Hint*: Run **heroku logs** at the command line.
3. Are you able to successfully update your password?

## 12.5 Conclusion

With the added password resets, our sample application's sign up, log in, and log out machinery is complete and professional-grade. The rest of the *Ruby on Rails Tutorial* builds on this foundation to make a site with Twitter-like micro-posts ([Chapter 13](#)) and a status feed of posts from followed users ([Chapter 14](#)).

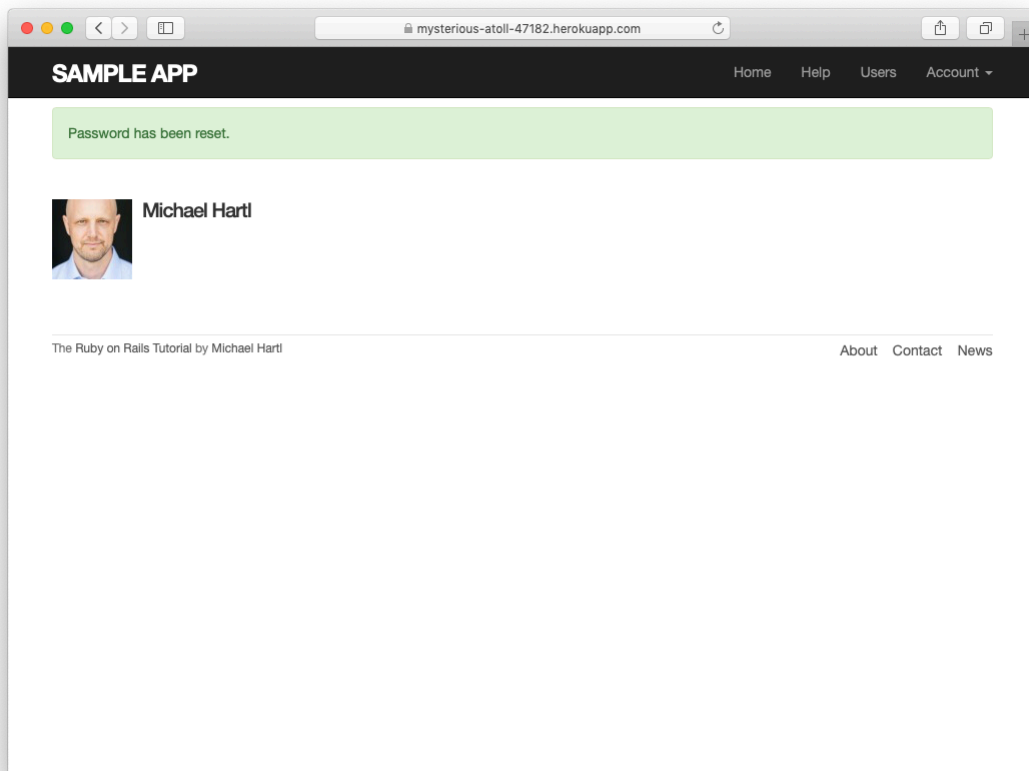


Figure 12.15: The result of a successful password reset in production.

In the process, we'll learn about some of the most powerful features of Rails, including image upload, custom database queries, and advanced data modeling with `has_many` and `has_many :through`.

### 12.5.1 What we learned in this chapter

- Like sessions and account activations, password resets can be modeled as a resource despite not being Active Record objects.
- Rails can generate Action Mailer actions and views to send email.
- Action Mailer supports both plain-text and HTML mail.
- As with ordinary actions and views, instance variables defined in mailer actions are available in mailer views.
- Password resets use a generated token to create a unique URL for resetting passwords.
- Password resets use a hashed reset digest to securely identify valid reset requests.
- Both mailer tests and integration tests are useful for verifying the behavior of the User mailer.
- We can send email in production using SendGrid.

## 12.6 Proof of expiration comparison

We saw in [Section 12.3](#) that the comparison test for determining when a password reset has expired is

```
reset_sent_at < 2.hours.ago
```

as seen in Listing 12.17. This looks like it might be read as “reset sent less than two hours ago”, which is the opposite of what we want. In this section, we’ll prove that the above comparison is correct.<sup>6</sup>

We start by defining two time intervals. Let  $\Delta t_r$  be the time interval since sending the password reset and  $\Delta t_e$  be the expiration time limit (e.g., two hours). A password reset has expired if the time interval since the reset was sent is greater than the expiration limit:

$$\Delta t_r > \Delta t_e. \quad (12.1)$$

If we write the time now as  $t_N$ , the password reset sending time as  $t_r$ , and the expiration time as  $t_e$  (e.g., two hours ago), then we have

$$\Delta t_r = t_N - t_r \quad (12.2)$$

and

$$\Delta t_e = t_N - t_e. \quad (12.3)$$

Plugging Eq. (12.2) and Eq. (12.3) into (12.1) then gives

$$\begin{aligned} \Delta t_r &> \Delta t_e \\ t_N - t_r &> t_N - t_e \\ -t_r &> -t_e, \end{aligned}$$

which upon multiplying through by  $-1$  yields

$$t_r < t_e. \quad (12.4)$$

Converting (12.4) to code with the value  $t_e = 2$  hours ago gives the **password\_reset\_expired?** method shown in Listing 12.17:

```
def password_reset_expired?
  reset_sent_at < 2.hours.ago
end
```

As noted in Section 12.3, if we read `<` as “earlier than” instead of “less than”, this code makes sense as the English sentence “The password reset was sent earlier than two hours ago.”

<sup>6</sup>This proof is the price you pay for reading a web development tutorial written by a Ph.D. physicist. Just be grateful I couldn’t find a way to work  $\left(-\frac{\hbar^2}{2m}\nabla^2 + V\right)\psi = E\psi$  or  $G^{\mu\nu} = 8\pi T^{\mu\nu}$  ( $= 4\tau T^{\mu\nu}$ ) into the exposition.