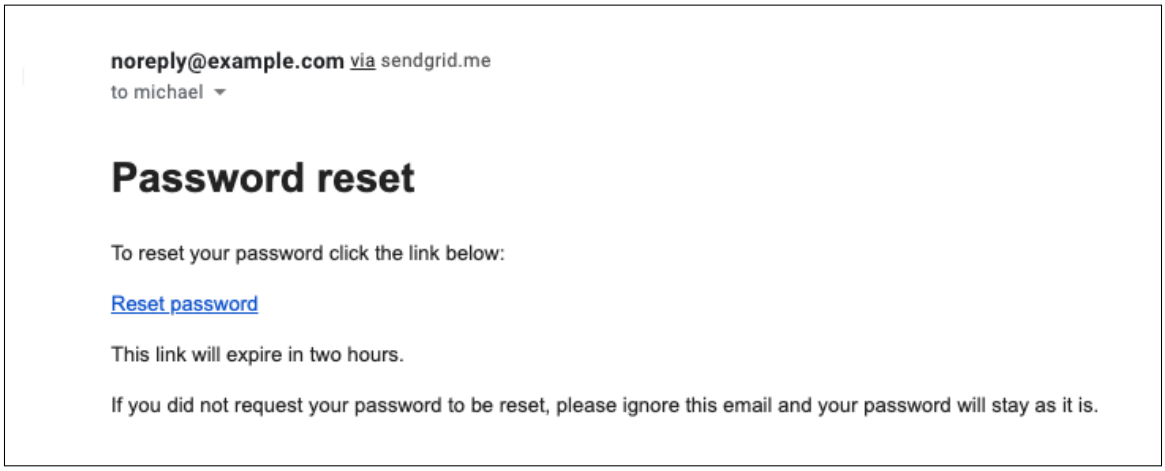

Figure 12.14: A password reset email sent in production.

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 microposts (Chapter 13) and a status feed of posts from followed users (Chapter 14).

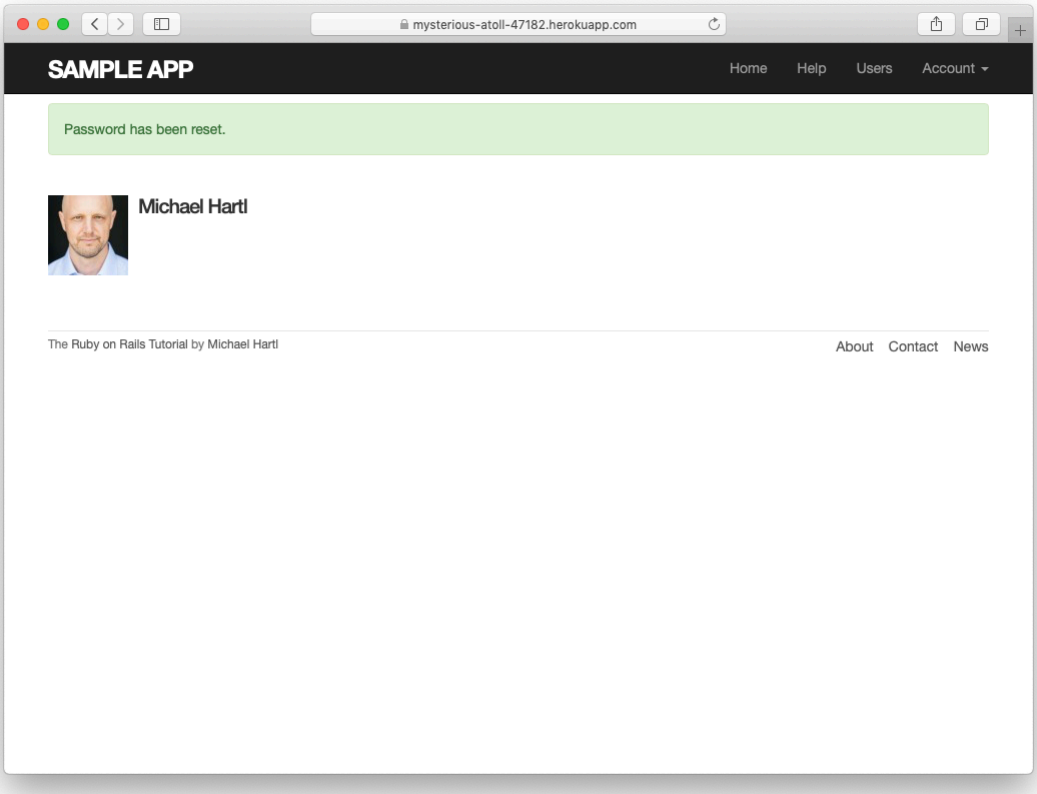

Figure 12.15: The result of a successful password reset in production.

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 likes 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.[6]

We start by defining two time intervals. Let Δt_r be the time interval since sending the password reset and Δt_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:

$$\Delta t_r > \Delta t_e. \tag{12.1}$$

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

$$\Delta t_r = t_N - t_r \tag{12.2}$$

and

$$\Delta t_e = t_N - t_e. \tag{12.3}$$

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

$$t_r < t_e. \tag{12.4}$$

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."

[6] 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.