Episode 274

Remember Me & Reset Password

Although there are a number of good available authentication solutions for Rails applications there's still a lot to be said for rolling your own. In episode 250 [watch[1], read[2]] we did exactly that and later, in episode 270 [watch[3], read[4]], we showed how Rails 3.1 makes it even easier by giving us `has_secure_password` which will automatically generate password hashes.

The authentication we implemented in those episodes was fairly basic so in this episode we'll add a couple of new features to make it better. First we'll add a "Remember Me" checkbox to the login page so that users can choose to be logged in automatically then we'll add a "Reset Password" link that will allow users who have forgotten their password to have it reset. We'll implement these features by extending the application we wrote in episode 270. This application uses Rails 3.1 but what we'll show here will work just as well in Rails 3.0.

Adding a "Remember Me" Checkbox

When a user logs in to our application their `id` is stored in the session. This takes place in the `SessionsController`'s `create` action.

/app/controllers/sessions_controller.rb

```
def create
  user = User.find_by_email(params[:email])
  if user && user.authenticate(params[:password])
    session[:user_id] = user.id
    redirect_to root_url, :notice => "Logged in!"
  else
    flash.now.alert = "Invalid email or password"
    render "new"
  end
end
```

When a logged-in user closes their browser the session cookie is deleted and they

---

[1] http://railscasts.com/episodes/250-authentication-from-scratch

[2] http://asciicasts.com/episodes/250-authentication-from-scratch

[3] http://railscasts.com/episodes/270-authentication-in-rails-3-1

[4] http://asciicasts.com/episodes/270-authentication-in-rails-3-1

need to log in again the next time they open the application. We'll replace this session cookie with a permanent one so that we can persist each user's `id`.

The most obvious problem with this is that the `id`s are stored as sequential integers. If the `id` is stored in a permanent cookie it would be easy for a malicious user to change the value and so view other users' data. To stop this we'll generate a unique token for each user that is unguessable and store that value in the cookie instead.

Each user will have their own token which will need to be stored in the database so we'll create a migration that will add an `auth_token` field to the `users` table and then migrate the database.

```
$ rails g migration add_auth_token_to_users auth_token:string
```

We need a way to generate this unique token when a user is created and so we'll write a method called `generate_token` in the `User` model. This method will take a `column` argument so that we can have multiple tokens later if need be.

/app/models/user.rb

```
class User < ActiveRecord::Base
  attr_accessible :email, :password, :password_confirmation
  has_secure_password
  validates_presence_of :password, :on => :create
  before_create { generate_token(:auth_token) }

  def generate_token(column)
    begin
      self[column] = SecureRandom.urlsafe_base64
    end while User.exists?(column => self[column])
  end
end
```

To create the token we use ActiveSupport's `SecureRandom`[5] class to generate a random string. We check that no other user exists with that token and repeatedly generate another random token while this is true. We call the method in a `before_create` filter so that the token is generated when a new user is saved for the first time. If we have pre-existing users in the database we'll need to create

[5] http://api.rubyonrails.org/classes/ActiveSupport/SecureRandom.html

tokens for them as we could create a rake task to do this, though we won't do so here.

We'll modify the `SessionsController`'s `create` action now so that when a user logs in we store their token in a cookie. We'll change the `destroy` action too so that the cookie is removed when a user logs out.

/app/controllers/sessions_controller.rb

```
class SessionsController < ApplicationController
  def new
  end

  def create
    user = User.find_by_email(params[:email])
    if user && user.authenticate(params[:password])
      cookies.permanent[:auth_token] = user.auth_token
      redirect_to root_url, :notice => "Logged in!"
    else
      flash.now.alert = "Invalid email or password"
      render "new"
    end
  end

  def destroy
    cookies.delete(:auth_token)
    redirect_to root_url, :notice => "Logged out!"
  end
end
```

Any user who logs in now is logged in permanently. They may not want this so we'll add a checkbox to the login form so that they can choose for themselves. The changes to the form are straightforward enough: we just need to add the checkbox along with a label that says what it's for.

/app/views/sessions/new.html.erb

```
<h1>Log in</h1>

<%= form_tag sessions_path do %>
  <div class="field">
    <%= label_tag :email %>
    <%= text_field_tag :email, params[:email] %>
  </div>
  <div class="field">
    <%= label_tag :password %>
    <%= password_field_tag :password %>
  </div>
  <div class="field">
    <%= label_tag :remember_me %>
    <%= check_box_tag :remember_me, 1, params[:remember_me] %>
  </div>
  <div class="actions"><%= submit_tag "Log in" %></div>
<% end %>
```

We can now modify the `SessionsController` so that the permanent cookie is only set if the user has checked the checkbox. If they haven't the login details will be stored in a session cookie.

/app/controllers/sessions_controller.rb

```
def create
  user = User.find_by_email(params[:email])
  if user && user.authenticate(params[:password])
    if params[:remember_me]
      cookies.permanent[:auth_token] = user.auth_token
    else
      cookies[:auth_token] = user.auth_token
    end
    redirect_to root_url, :notice => "Logged in!"
  else
    flash.now.alert = "Invalid email or password"
    render "new"
  end
end
```

There's one more change we need to make. The `ApplicationController` needs to

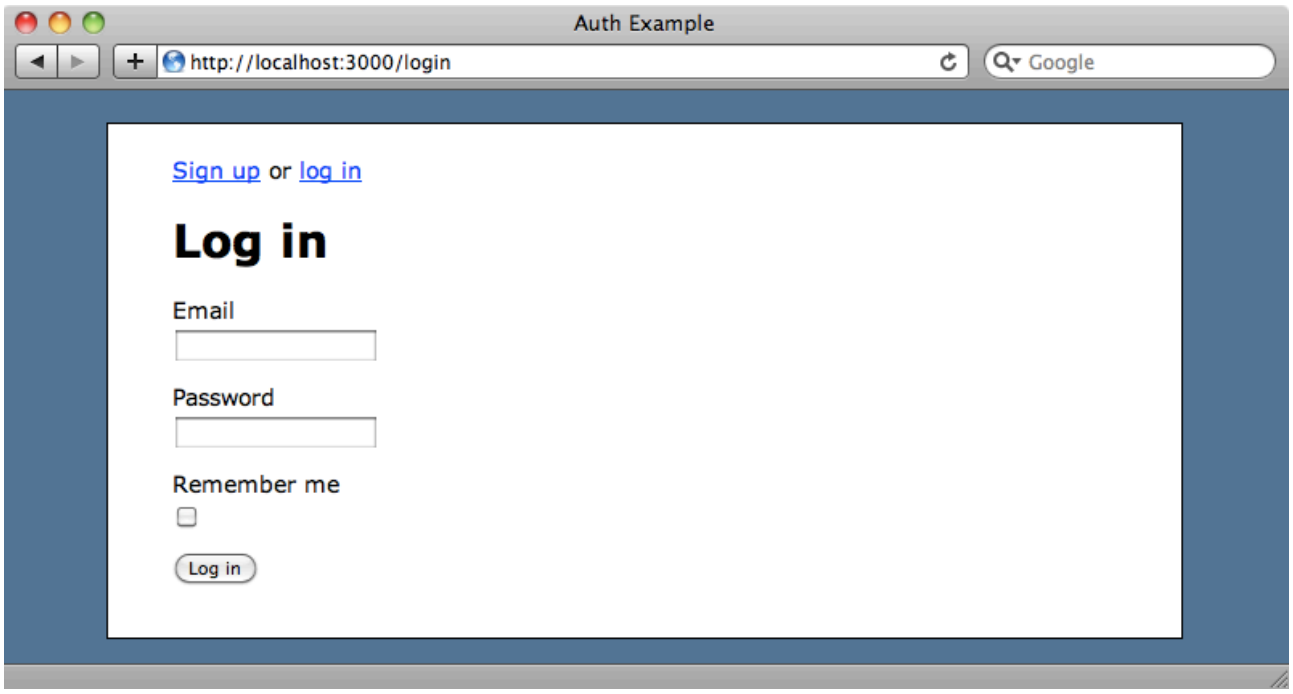
be changed so that it reads the authentication token from the cookie rather than a user's `id` from the session.

/app/controllers/application_controller.rb

```
class ApplicationController < ActionController::Base
  protect_from_forgery

  private
  def current_user
    @current_user ||= User.find_by_auth_token( ↵
      cookies[:auth_token]) if cookies[:auth_token]
  end
  helper_method :current_user
end
```

We can try this out now. When we log into our application now we'll see the "Remember me" checkbox. If we check the box when we log in and then close and reopen the browser we'll be logged in automatically. Our "remember me" functionality is working just as we want.



Adding “Forgotten Password” Functionality

Now we’ll take a look at allowing a user to reset a forgotten password. We’ll start by adding a suitable link in the login form.

/app/views/sessions/new.html.erb

```
<h1>Log in</h1>

<%= form_tag sessions_path do %>
  <div class="field">
    <%= label_tag :email %>
    <%= text_field_tag :email, params[:email] %>
  </div>
  <div class="field">
    <%= label_tag :password %>
    <%= password_field_tag :password %>
  </div>
  <p><%= link_to "forgotten password?", ↵
    new_password_reset_path %></p>
  <div class="field">
    <%= label_tag :remember_me %>
    <%= check_box_tag :remember_me, 1, params[:remember_me] %>
  </div>
  <div class="actions"><%= submit_tag "Log in" %></div>
<% end %>
```

The link points to `new_password_reset_path` which is part of a resource that we haven’t yet written. We’ll fix that now by creating a `PasswordResets` controller with a `new` action.

```
$ rails g controller password_resets new
```

We want to treat this controller as a resource so we’ll modify the routes file, replacing the generated route with a call to `resources`.

/config/routes.rb

```
Auth::Application.routes.draw do

  get "logout" => "sessions#destroy", :as => "logout"
  get "login" => "sessions#new", :as => "login"
  get "signup" => "users#new", :as => "signup"
  root :to => "home#index"
  resources :users
  resources :sessions
  resources :password_resets
end
```

This isn't a proper model-backed resource but it will work for us.

In the new action's view we'll create a form to allow a user to enter their email address and request that their password is reset. The form looks like this:

/app/views/password_resets/new.html.erb

```
<h1>Reset Password</h1>

<%= form_tag password_resets_path, :method => :post do %>
  <div class="field">
    <%= label_tag :email %>
    <%= text_field_tag :email, params[:email] %>
  </div>
  <div class="actions"><%= submit_tag "Reset Password" %></div>
<% end %>
```

This isn't a model-backed resource so we're using `form_tag` here. The form POSTs to the `PasswordResets` controller's `create` action and we'll write that action next. In it we'll find the `User` with the supplied email address and send them instructions on how to reset their password. This will be done in a new `send_password_reset` method in the `User` model.

/app/controllers/password_resets.rb

```
def create
  user = User.find_by_email(params[:email])
  user.send_password_reset if user
  redirect_to root_url, :notice => "Email sent with ↵
    password reset instructions."
end
```

The notice is shown whether the user is found or not. This makes things a little more secure so that a malicious user can't determine whether a given user exists in the database.

We'll write the `send_password_reset` method now. In it we'll send an email containing a token for the password reset request. We want the token to expire after a given period, say a couple of hours, so that the link is valid only for a short time after the reset is requested. To hold this data we'll need a couple of extra fields in the `users` table so we'll write and run a migration to create these.

```
$ rails g migration add_password_reset_to_users ↵
  password_reset_token:string password_reset_sent_at:datetime
```

In `send_password_reset` we'll use the `generate_token` method we wrote earlier to create a password reset token. We'll also set the `password_reset_sent_at` field so that we know when the token should expire and then save the `User` so that these details are stored. After we've saved the changes to the `User` we'll pass it to a `UserMailer` so that we can sent the reset email.

/app/models/user.rb

```
def send_password_reset
  generate_token(:password_reset_token)
  self.password_reset_sent_at = Time.zone.now
  save!
  UserMailer.password_reset(self).deliver
end
```

We haven't created the `UserMailer` yet so we'll do that next.

```
$ rails g mailer user_mailer password_reset
```

In the mailer we'll assign the user to an instance variable so that we can access it from the template and set the recipient and subject.

/app/mailers/user_mailer.rb

```
class UserMailer < ActionMailer::Base
  default from: "from@example.com"

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

In the template we'll write some instructions and provide a link to reset the password.

/app/views/user_mailer/password_reset_text.erb

```
To reset your password click the URL below.

<%= edit_password_reset_url(@user.password_reset_token) %>

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

The link in the email sends the user to the `PasswordResetsController`'s `edit` action. Technically this isn't the best RESTful approach but it will work well enough for us. To get URLs working in mailers we'll need to alter our environment configuration and add the following line to `development.rb`.

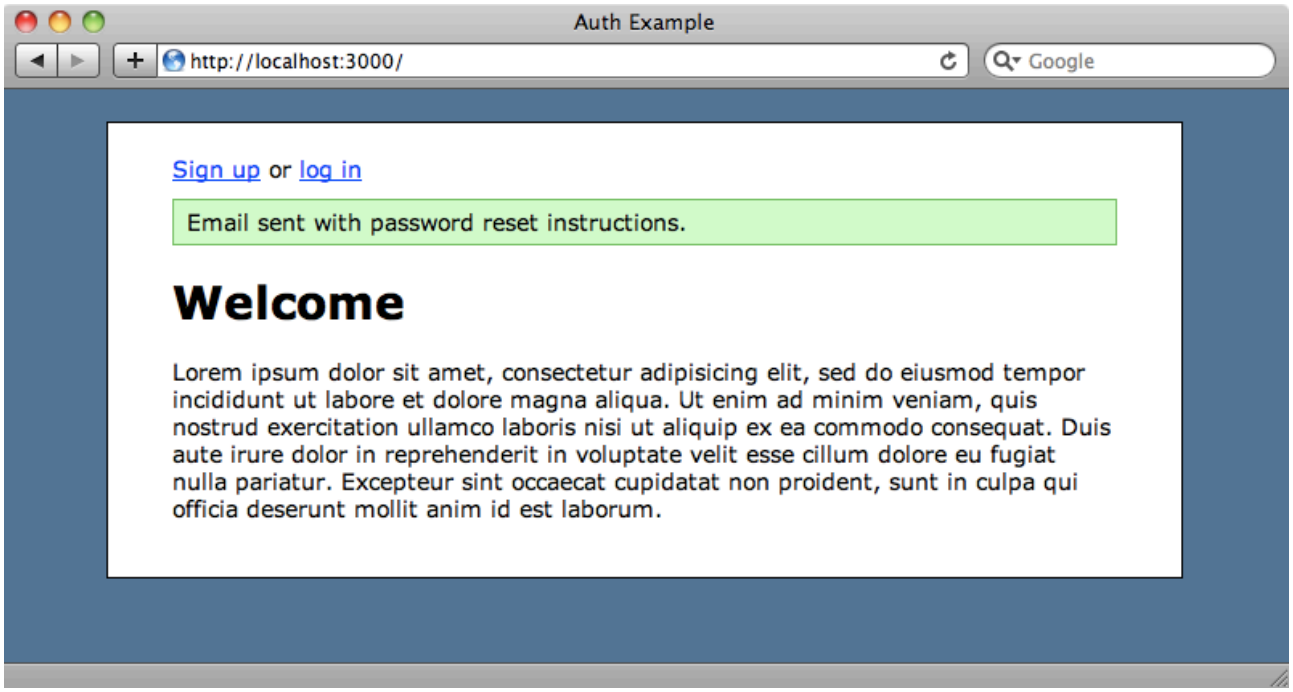
/config/environments/development.rb

```
Auth::Application.configure do
  # Other config items omitted.

  config.action_mailer.default_url_options = { :host => ↵
    "localhost:3000" }
end
```

We'll add a similar line in `production.rb` with the live domain name.

Let's try this out. If we visit the password reset page and enter our email address we should be told that an email containing the reset instructions has been sent.



When we check the development log we'll see the details of the email.

```
Sent mail to eifion@asciicasts.com (65ms)
Date: Thu, 14 Jul 2011 20:18:48 +0100
From: from@example.com
To: eifion@asciicasts.com
Message-ID: <4e1f4118af661_31a81639e544652a@noonoo.home.mail>
Subject: Password Reset
Mime-Version: 1.0
Content-Type: text/plain;
 charset=UTF-8
Content-Transfer-Encoding: 7bit

To reset your password click the URL below.

http://localhost:3000/password_resets/DeStUAsv2QTX_SR3ub_N0g/edit

If you did not request your password to be reset please ignore
this email and your password will stay as it is.
Redirected to http://localhost:3000/
Completed 302 Found in 1889ms
```

The email includes a link that contains the password reset URL. This URL includes the reset token as the `id` parameter.

Next we have to write the `edit` action. In it we'll fetch the user by their reset token. Note that we use the method with an exclamation mark so that if the user isn't found a 404 error is thrown.

/app/controllers/password_resets_controller.rb

```
def edit
  @user = User.find_by_password_reset_token!(params[:id])
end
```

In the associated view we'll create a form to allow the user to reset their password.

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

```
<h1>Reset Password</h1>

<%= form_for @user, :url => password_reset_path(params[:id]) ↵
  do |f| %>
  <% if @user.errors.any? %>
    <div class="error_messages">
      <h2>Form is invalid</h2>
      <ul>
        <% for message in @user.errors.full_messages %>
          <li><%= message %></li>
        <% end %>
      </ul>
    </div>
  <% end %>
  <div class="field">
    <%= f.label :password %>
    <%= f.password_field :password %>
  </div>
  <div class="field">
    <%= f.label :password_confirmation %>
    <%= f.password_field :password_confirmation %>
  </div>
  <div class="actions"><%= f.submit "Update Password" %></div>
<% end %>
```

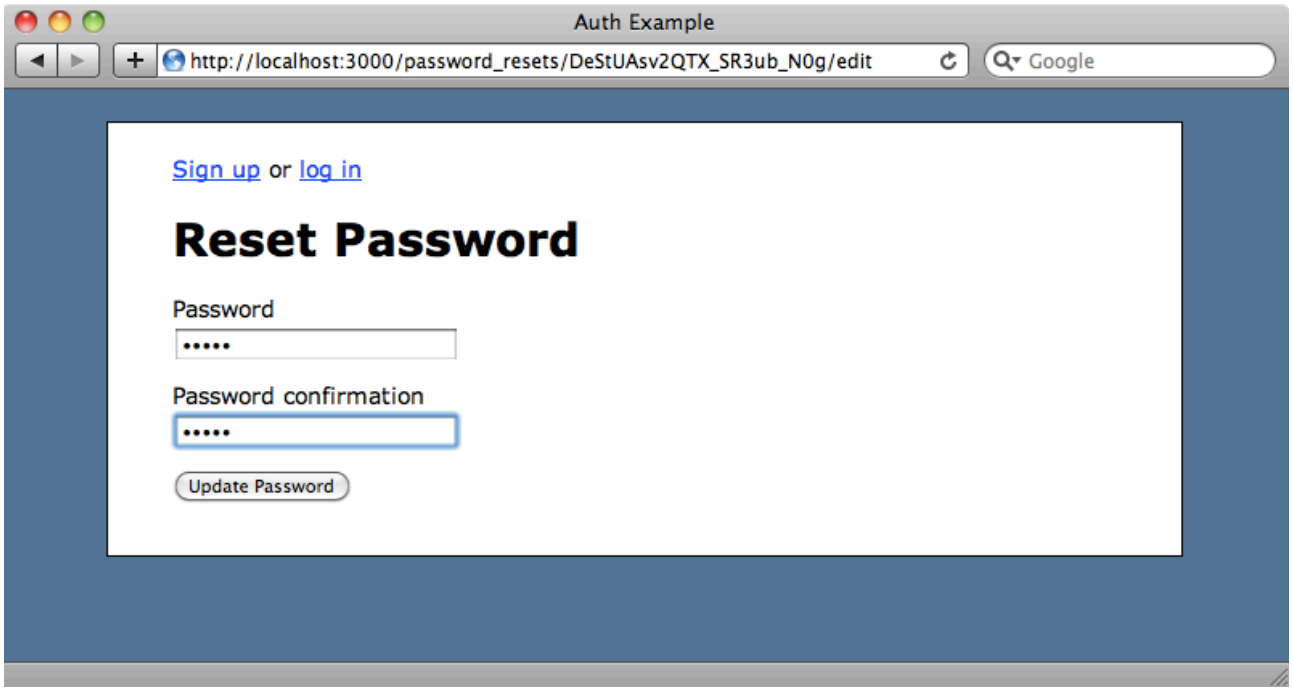
We use `form_for` in this form as we're modifying a resource. Because of this we have to explicitly set the `:url` parameter so that the form isn't POSTed to the `UsersController`. Instead it is sent to the `PasswordResetsController`'s `update` action, passing in the reset token as the `id`. The form contains a section for displaying any error messages and fields for entering and confirming the new password.

We'll write the `update` action next. This first checks that the password reset token is less than two hours old; if it isn't then we redirect the user to the reset form again. If the token is recent enough we then try to update the user. If this is successful we redirect to the home page and show a message; if not there must have been an error in the form so we render it again.

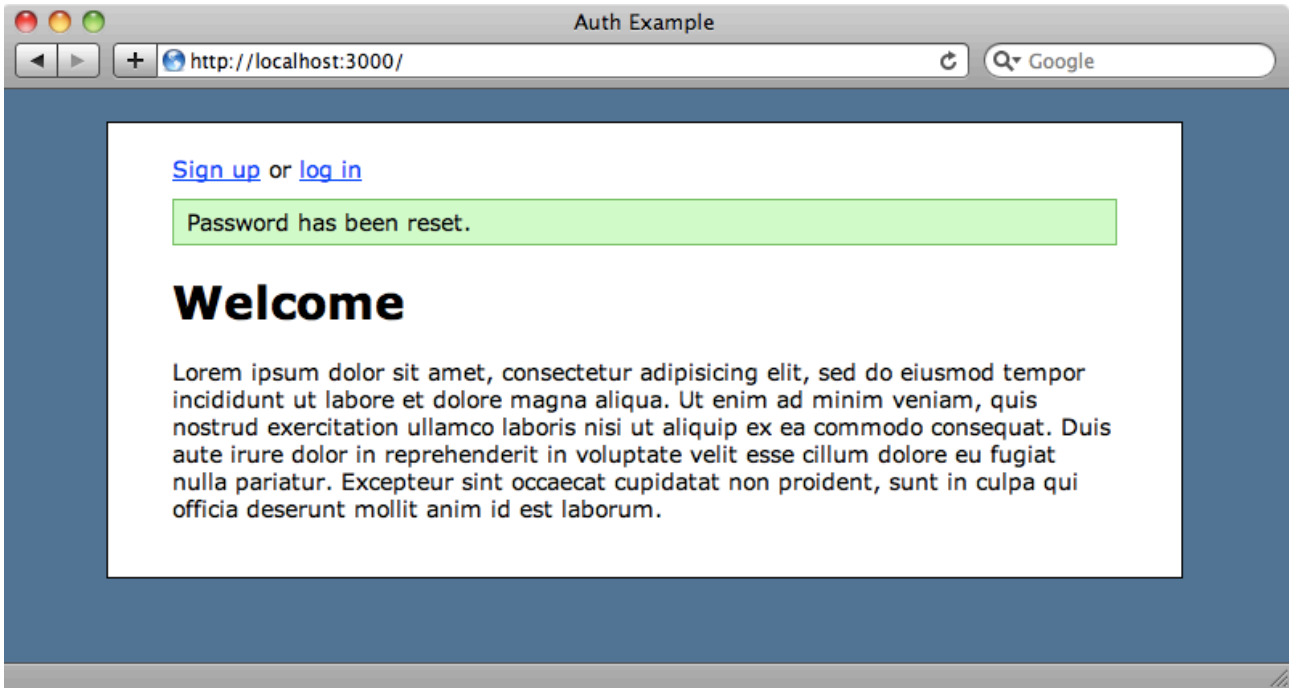
/app/controllers/password_resets_controller.rb

```
def update
  @user = User.find_by_password_reset_token!(params[:id])
  if @user.password_reset_sent_at < 2.hours.ago
    redirect_to new_password_reset_path, :alert => "Password ↵
      reset has expired."
  elsif @user.update_attributes(params[:user])
    redirect_to root_url, :notice => "Password has been reset."
  else
    render :edit
  end
end
```

We can try this out by pasting the URL from the reset email into the browser.



If we enter non-matching passwords we'll see an error message but when we submit the form correctly our password is successfully reset.



We can use this password reset idea to add other features, for example to confirm new account registrations. This is very similar to password resets but instead of resetting the password when the link is clicked we set a flag in the database to say that the account is confirmed as registered.

That’s it for this episode on remembering logins and resetting passwords. Some tools, such as Devise, provide this functionality but it can be useful to do it from scratch especially if you’re going to need to do a lot of customization.