## 11.1 Account activations resource

As with sessions (Section 8.1), we'll model account activations as a resource even though they won't be associated with an Active Record model. Instead, we'll include the relevant data (including the activation token and activation status) in the User model itself.

Because we'll be treating account activations as a resource, we'll interact with them via a standard REST URL. The activation link will be modifying the user's activation status, and for such modifications the standard REST practice is to issue a `PATCH` request to the **`update`** action (Table 7.1). The activation link needs to be sent in an email, though, and hence will involve a regular browser click, which issues a `GET` request instead of `PATCH`. This design constraint means that we can't use the **`update`** action, but we'll do the next-best thing and use the **`edit`** action instead, which does respond to `GET` requests.

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

```
$ git checkout -b account-activation
```

### 11.1.1 Account activations controller

As with Users and Sessions, the actions (or, in this case, the sole action) for the Account Activations resource will live inside an Account Activations controller, which we can generate as follows:[4]

```
$ rails generate controller AccountActivations
```

As we'll see in Section 11.2.1, the activation email will involve a URL of the form

[4]Because we'll be using an **`edit`** action, we could include **`edit`** on the command line, but this would also generate both an edit view and a test, neither of which we'll turn out to need.

| HTTP request | URL | Action | Named route |
|---|---|---|---|
| GET | /account_activation/<token>/edit | edit | edit_account_activation_url(token) |

Table 11.2: RESTful route provided by the Account Activations resource in Listing 11.1.

```
edit_account_activation_url(activation_token, ...)
```

which means we'll need a named route for the **edit** action. We can arrange for this with the **resources** line shown in Listing 11.1, which gives the RESTful route shown in Table 11.2.

**Listing 11.1:** Adding a route for the Account Activations **edit** action.
*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]
end
```

We'll define the **edit** action itself in Section 11.3.2, after we've finished the Account Activations data model and mailers.

### 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 11.2 list the **_url** form of the named route instead of the **_path** form? *Hint*: We're going to use it in an email.

### 11.1.2 Account activation data model

As discussed in the introduction, we need a unique activation token for use in the activation email. One possibility would be to use a string that's stored both in the database and included in the activation URL, but this raises security concerns if our database is compromised. For example, an attacker with access to the database could immediately activate newly created accounts (thereby logging in as the user), and could then change the password to gain control.[5]

To prevent such scenarios, we'll follow the example of passwords (Chapter 6) and remember tokens (Chapter 9) by pairing a publicly exposed virtual attribute with a secure hash digest saved to the database. This way we can access the activation token using

```
user.activation_token
```

and authenticate the user with code like

```
user.authenticated?(:activation, token)
```

(This will require a modification of the **authenticated?** method defined in Listing 9.6.)

We'll also add a boolean attribute called **activated** to the User model, which will allow us to test if a user is activated using the same kind of autogenerated boolean method we saw in Section 10.4.1:

```
if user.activated? ...
```

[5]It's mainly for this reason that we won't be using the (perhaps slightly misnamed) **has_secure_token** facility added in Rails 5, which stores the corresponding token in the database as unhashed cleartext.

| users | |
|---|---|
| id | integer |
| name | string |
| email | string |
| created_at | datetime |
| updated_at | datetime |
| password_digest | string |
| remember_digest | string |
| admin | boolean |
| activation_digest | string |
| activated | boolean |
| activated_at | datetime |

Figure 11.1: The User model with added account activation attributes.

Finally, although we won't use it in this tutorial, we'll record the time and date of the activation in case we want it for future reference. The full data model appears in Figure 11.1.

The migration to add the data model from Figure 11.1 adds all three attributes at the command line:

```
$ rails generate migration add_activation_to_users \
> activation_digest:string activated:boolean activated_at:datetime
```

(Here the `>` on the second line is a "line continuation" character inserted automatically by the shell, and should not be typed literally.) As with the `admin` attribute (Listing 10.54), we'll add a default boolean value of `false` to the `activated` attribute, as shown in Listing 11.2.

**Listing 11.2:** A migration for account activation (with added index).
*db/migrate/[timestamp]_add_activation_to_users.rb*

```
class AddActivationToUsers < ActiveRecord::Migration[6.0]
  def change
    add_column :users, :activation_digest, :string
    add_column :users, :activated, :boolean, default: false
    add_column :users, :activated_at, :datetime
  end
end
```

We then apply the migration as usual:

```
$ rails db:migrate
```

### Activation token callback

Because every newly signed-up user will require activation, we should assign an activation token and digest to each user object before it's created. We saw a similar idea in Section 6.2.5, where we needed to convert an email address to lower-case before saving a user to the database. In that case, we used a **before_save** callback combined with the **downcase** method (Listing 6.32). A **before_save** callback is automatically called before the object is saved, which includes both object creation and updates, but in the case of the activation digest we only want the callback to fire when the user is created. This requires a **before_create** callback, which we'll define as follows:

```
before_create :create_activation_digest
```

This code, called a *method reference*, arranges for Rails to look for a method called **create_activation_digest** and run it before creating the user. (In Listing 6.32, we passed **before_save** an explicit block, but the method reference technique is generally preferred.) Because the **create_activation_digest** method itself is only used internally by the User model, there's no

need to expose it to outside users; as we saw in Section 7.3.2, the Ruby way to accomplish this is to use the **private** keyword:

```
private

  def create_activation_digest
    # Create the token and digest.
  end
```

All methods defined in a class after **private** are automatically hidden, as seen in this console session:

```
$ rails console
>> User.first.create_activation_digest
NoMethodError: private method `create_activation_digest' called for #<User>
```

The purpose of the **before_create** callback is to assign the token and corresponding digest, which we can accomplish as follows:

```
self.activation_token  = User.new_token
self.activation_digest = User.digest(activation_token)
```

This code simply reuses the token and digest methods used for the remember token, as we can see by comparing it with the **remember** method from Listing 9.3:

```
# Remembers a user in the database for use in persistent sessions.
def remember
  self.remember_token = User.new_token
  update_attribute(:remember_digest, User.digest(remember_token))
end
```

The main difference is the use of **update_attribute** in the latter case. The reason for the difference is that remember tokens and digests are created for users that already exist in the database, whereas the **before_create** callback happens *before* the user has been created, so there's not yet any attribute to

update. As a result of the callback, when a new user is defined with **`User.new`** (as in user signup, Listing 7.19), it will automatically get both **`activation_token`** and **`activation_digest`** attributes; because the latter is associated with a column in the database (Figure 11.1), it will be written to the database automatically when the user is saved.

Putting together the discussion above yields the User model shown in Listing 11.3. As required by the virtual nature of the activation token, we've added a second **`attr_accessor`** to our model. Note that we've taken the opportunity to replace the email downcasing callback from Listing 6.32 with a method reference.

**Listing 11.3:** Adding account activation code to the User model. GREEN
*app/models/user.rb*

```
class User < ApplicationRecord
  attr_accessor :remember_token, :activation_token
  before_save   :downcase_email
  before_create :create_activation_digest
  validates :name,  presence: true, length: { maximum: 50 }
  .
  .
  .
  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
```

### Seed and fixture users

Before moving on, we should also update our seed data and fixtures so that our sample and test users are initially activated, as shown in Listing 11.4 and Listing 11.5. (The **`Time.zone.now`** method is a built-in Rails helper that returns

the current timestamp, taking into account the time zone on the server.)

**Listing 11.4:** Activating seed users by default.
*db/seeds.rb*

```
# Create a main sample user.
User.create!(name:  "Example User",
             email: "example@railstutorial.org",
             password:              "foobar",
             password_confirmation: "foobar",
             admin:     true,
             activated: true,
             activated_at: Time.zone.now)

# Generate a bunch of additional users.
99.times do |n|
  name  = Faker::Name.name
  email = "example-#{n+1}@railstutorial.org"
  password = "password"
  User.create!(name:  name,
               email: email,
               password:              password,
               password_confirmation: password,
               activated: true,
               activated_at: Time.zone.now)
end
```

**Listing 11.5:** Activating fixture users.
*test/fixtures/users.yml*

```
michael:
  name: Michael Example
  email: michael@example.com
  password_digest: <%= User.digest('password') %>
  admin: true
  activated: true
  activated_at: <%= Time.zone.now %>

archer:
  name: Sterling Archer
  email: duchess@example.gov
  password_digest: <%= User.digest('password') %>
  activated: true
  activated_at: <%= Time.zone.now %>

lana:
  name: Lana Kane
```

```
  email: hands@example.gov
  password_digest: <%= User.digest('password') %>
  activated: true
  activated_at: <%= Time.zone.now %>

malory:
  name: Malory Archer
  email: boss@example.gov
  password_digest: <%= User.digest('password') %>
  activated: true
  activated_at: <%= Time.zone.now %>

<% 30.times do |n| %>
user_<%= n %>:
  name:  <%= "User #{n}" %>
  email: <%= "user-#{n}@example.com" %>
  password_digest: <%= User.digest('password') %>
  activated: true
  activated_at: <%= Time.zone.now %>
<% end %>
```

To apply the changes in Listing 11.4, reset the database to reseed the data as usual:

```
$ rails db:migrate:reset
$ rails db:seed
```

### 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 after the changes made in this section.

2. By instantiating a User object in the console, confirm that calling the **`create_activation_digest`** method raises a **`NoMethodError`** due to its being a private method. What is the value of the user's activation digest?