up or act strangely when you're using the debugger; use your technical sophistication (Box 1.2) to resolve any issues.)

## 10.3 Showing all users

In this section, we'll add the penultimate user action, the **`index`** action, which is designed to display *all* the users instead of just one. Along the way, we'll learn how to seed the database with sample users and how to *paginate* the user output so that the index page can scale up to display a potentially large number of users. A mockup of the result—users, pagination links, and a "Users" navigation link—appears in Figure 10.8.[8] In Section 10.4, we'll add an administrative interface to the users index so that users can also be destroyed.

### 10.3.1 Users index

To get started with the users index, we'll first implement a security model. Although we'll keep individual user **`show`** pages visible to all site visitors, the user **`index`** will be restricted to logged-in users so that there's a limit to how much unregistered users can see by default.[9]

To protect the **`index`** page from unauthorized access, we'll first add a short test to verify that the **`index`** action is redirected properly (Listing 10.34).

**Listing 10.34:** Testing the **`index`** action redirect. RED
*test/controllers/users_controller_test.rb*

```
require 'test_helper'

class UsersControllerTest < ActionDispatch::IntegrationTest

  def setup
    @user       = users(:michael)
    @other_user = users(:archer)
  end
```

[8]Image retrieved from https://www.flickr.com/photos/glasgows/338937124/ on 2014-08-25. Copyright © 2008 by M&R Glasgow and used unaltered under the terms of the Creative Commons Attribution 2.0 Generic license.

[9]This is the same authorization model used by Twitter.

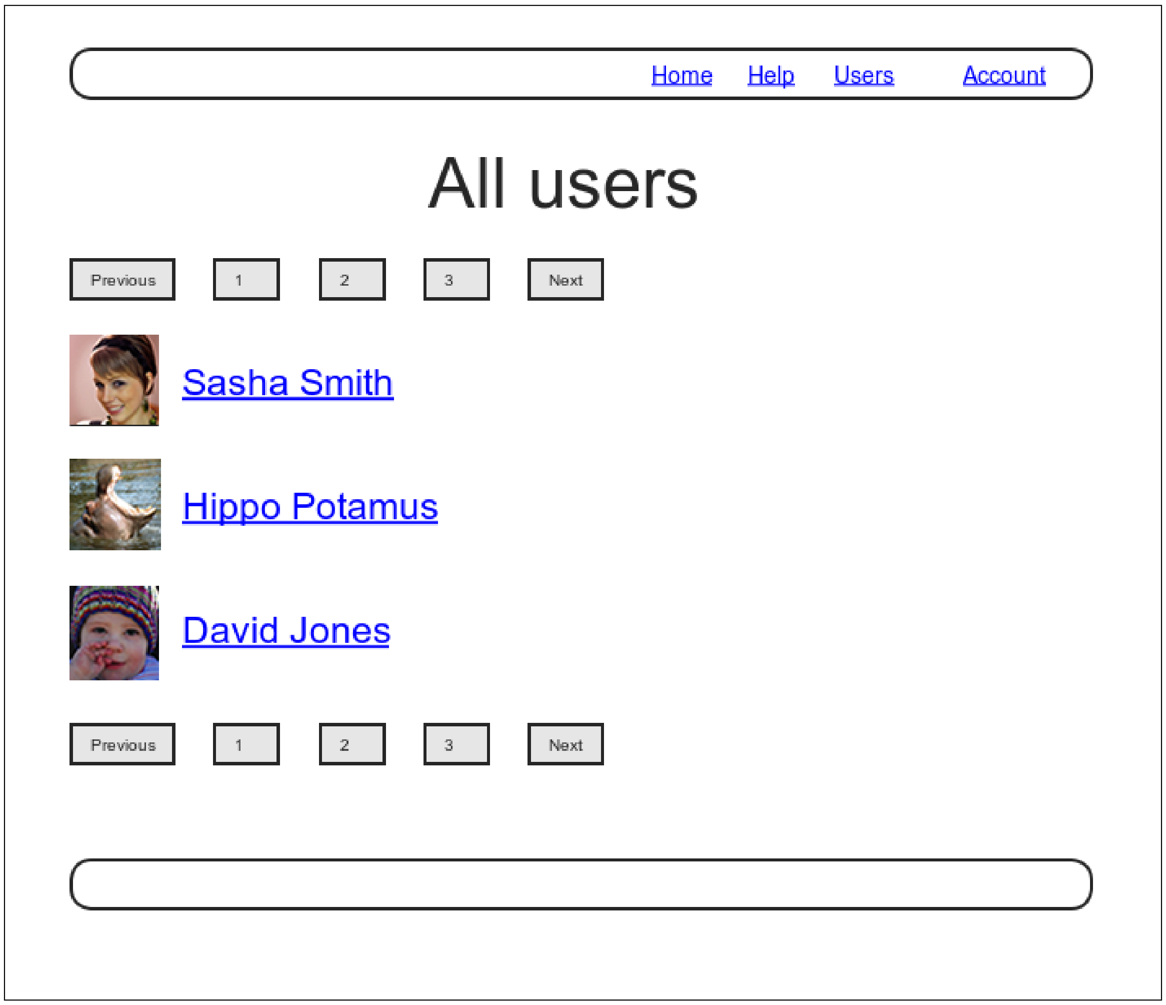


Figure 10.8: A mockup of the users index page.

```
  test "should get new" do
    get signup_path
    assert_response :success
  end

  test "should redirect index when not logged in" do
    get users_path
    assert_redirected_to login_url
  end
  .
  .
  .
end
```

Then we just need to add an **`index`** action and include it in the list of actions protected by the **`logged_in_user`** before filter (Listing 10.35).

**Listing 10.35:** Requiring a logged-in user for the **`index`** action. GREEN
*app/controllers/users_controller.rb*

```
class UsersController < ApplicationController
  before_action :logged_in_user, only: [:index, :edit, :update]
  before_action :correct_user,   only: [:edit, :update]

  def index
  end

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

To display the users themselves, we need to make a variable containing all the site's users and then render each one by iterating through them in the index view. As you may recall from the corresponding action in the toy app (Listing 2.9), we can use **`User.all`** to pull all the users out of the database, assigning them to an **`@users`** instance variable for use in the view, as seen in Listing 10.36. (If displaying all the users at once seems like a bad idea, you're right, and we'll remove this blemish in Section 10.3.3.)

**Listing 10.36:** The user **index** action.
*app/controllers/users_controller.rb*

```
class UsersController < ApplicationController
  before_action :logged_in_user, only: [:index, :edit, :update]
  .
  .
  .
  def index
    @users = User.all
  end
  .
  .
  .
end
```

To make the actual index page, we'll make a view (which you'll have to create) that iterates through the users and wraps each one in an **li** tag. We do this with the **each** method, displaying each user's Gravatar and name, while wrapping the whole thing in a **ul** tag (Listing 10.37).

**Listing 10.37:** The users index view.
*app/views/users/index.html.erb*

```
<% provide(:title, 'All users') %>
<h1>All users</h1>

<ul class="users">
  <% @users.each do |user| %>
    <li>
      <%= gravatar_for user, size: 50 %>
      <%= link_to user.name, user %>
    </li>
  <% end %>
</ul>
```

The code in Listing 10.37 uses the result of Listing 10.38 from Section 7.1.4, which allows us to pass an option to the Gravatar helper specifying a size other than the default. If you didn't do that exercise, update your Users helper file with the contents of Listing 10.38 before proceeding. (You are also welcome to use the Ruby 2.0–style version from Listing 7.13 instead.)

**Listing 10.38:** Adding an options hash in the **gravatar_for** helper.
*app/helpers/users_helper.rb*

```
module UsersHelper

  # Returns the Gravatar for the given user.
  def gravatar_for(user, options = { size: 80 })
    size         = options[:size]
    gravatar_id  = Digest::MD5::hexdigest(user.email.downcase)
    gravatar_url = "https://secure.gravatar.com/avatar/#{gravatar_id}?s=#{size}"
    image_tag(gravatar_url, alt: user.name, class: "gravatar")
  end
end
```

Let's also add a little CSS (or, rather, SCSS) for style (Listing 10.39).

**Listing 10.39:** CSS for the users index.
*app/assets/stylesheets/custom.scss*

```
.
.
.
/* Users index */

.users {
  list-style: none;
  margin: 0;
  li {
    overflow: auto;
    padding: 10px 0;
    border-bottom: 1px solid $gray-lighter;
  }
}
```

Finally, we'll add the URL to the users link in the site's navigation header using **users_path**, thereby using the last of the unused named routes in Table 7.1. The result appears in Listing 10.40.

**Listing 10.40:** Adding the URL to the users link.
*app/views/layouts/_header.html.erb*

```
<header class="navbar navbar-fixed-top navbar-inverse">
  <div class="container">
```

```
    <%= link_to "sample app", root_path, id: "logo" %>
    <nav>
      <ul class="nav navbar-nav navbar-right">
        <li><%= link_to "Home", root_path %></li>
        <li><%= link_to "Help", help_path %></li>
        <% if logged_in? %>
          <li><%= link_to "Users", users_path %></li>
          <li class="dropdown">
            <a href="#" class="dropdown-toggle" data-toggle="dropdown">
              Account <b class="caret"></b>
            </a>
            <ul class="dropdown-menu">
              <li><%= link_to "Profile", current_user %></li>
              <li><%= link_to "Settings", edit_user_path(current_user) %></li>
              <li class="divider"></li>
              <li>
                <%= link_to "Log out", logout_path, method: :delete %>
              </li>
            </ul>
          </li>
        <% else %>
          <li><%= link_to "Log in", login_path %></li>
        <% end %>
      </ul>
    </nav>
  </div>
</header>
```

With that, the users index is fully functional, with all tests GREEN:

**Listing 10.41:** GREEN

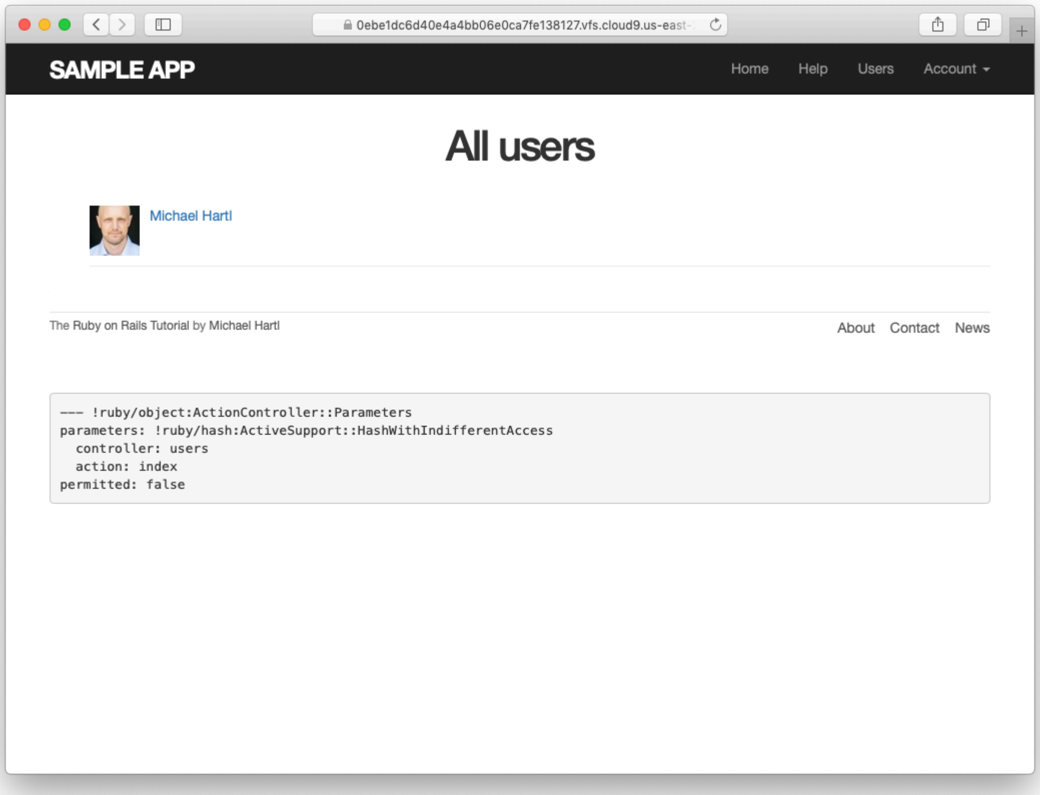
```
$ rails test
```

On the other hand, as seen in Figure 10.9, it is a bit… lonely. Let's remedy this sad situation.

### 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.



All users

Michael Hartl

The Ruby on Rails Tutorial by Michael Hartl

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

Figure 10.9: The users index page with only one user.

1. We've now filled in all the links in the site layout. Write an integration test for all the layout links, including the proper behavior for logged-in and non-logged-in users. *Hint*: Use the **log_in_as** helper and add to the steps shown in Listing 5.32.

### 10.3.2 Sample users

In this section, we'll give our lonely sample user some company. Of course, to create enough users to make a decent users index, we *could* use our web browser to visit the signup page and make the new users one by one, but a far better solution is to use Ruby to make the users for us.

First, we'll add the *Faker* gem to the **Gemfile**, which will allow us to make sample users with semi-realistic names and email addresses (Listing 10.42).[10] (Ordinarily, you'd probably want to restrict the `faker` gem to a development environment, but in the case of the sample app we'll be using it on our production site as well (Section 10.5).)

**Listing 10.42:** Adding the Faker gem to the **Gemfile**.

```
source 'https://rubygems.org'

gem 'rails',          '6.0.1'
gem 'bcrypt',         '3.1.13'
gem 'faker',          '2.1.2'
gem 'bootstrap-sass', '3.4.1'
.
.
.
```

Then install as usual:

```
$ bundle install
```

Next, we'll add a Ruby program to seed the database with sample users, for which Rails uses the standard file **db/seeds.rb**. The result appears in

[10]As always, you should use the version numbers listed at gemfiles-6th-ed.railstutorial.org instead of the ones listed here.

Listing 10.43. (The code in Listing 10.43 is a bit advanced, so don't worry too much about the details.)

**Listing 10.43:** A program for seeding the database with sample users.
*db/seeds.rb*

```
# Create a main sample user.
User.create!(name:  "Example User",
             email: "example@railstutorial.org",
             password:              "foobar",
             password_confirmation: "foobar")

# 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)
end
```

The code in Listing 10.43 creates an example user with name and email address replicating our previous one, and then makes 99 more. The **`create!`** method is just like the **`create`** method, except it raises an exception (Section 6.1.4) for an invalid user rather than returning **`false`**. This behavior makes debugging easier by avoiding silent errors.

With the code as in Listing 10.43, we can reset the database and then invoke the Rake task using **`db:seed`**:[11]

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

Seeding the database can be slow, and on some systems could take up to a few minutes. Also, some readers have reported that they are unable to run the reset command if the Rails server is running, so you may have to stop the server first before proceeding (Box 1.2).

[11]In principle, these two tasks can be combined in **`rails db:reset`**, but as of this writing this command doesn't work with the latest version of Rails.

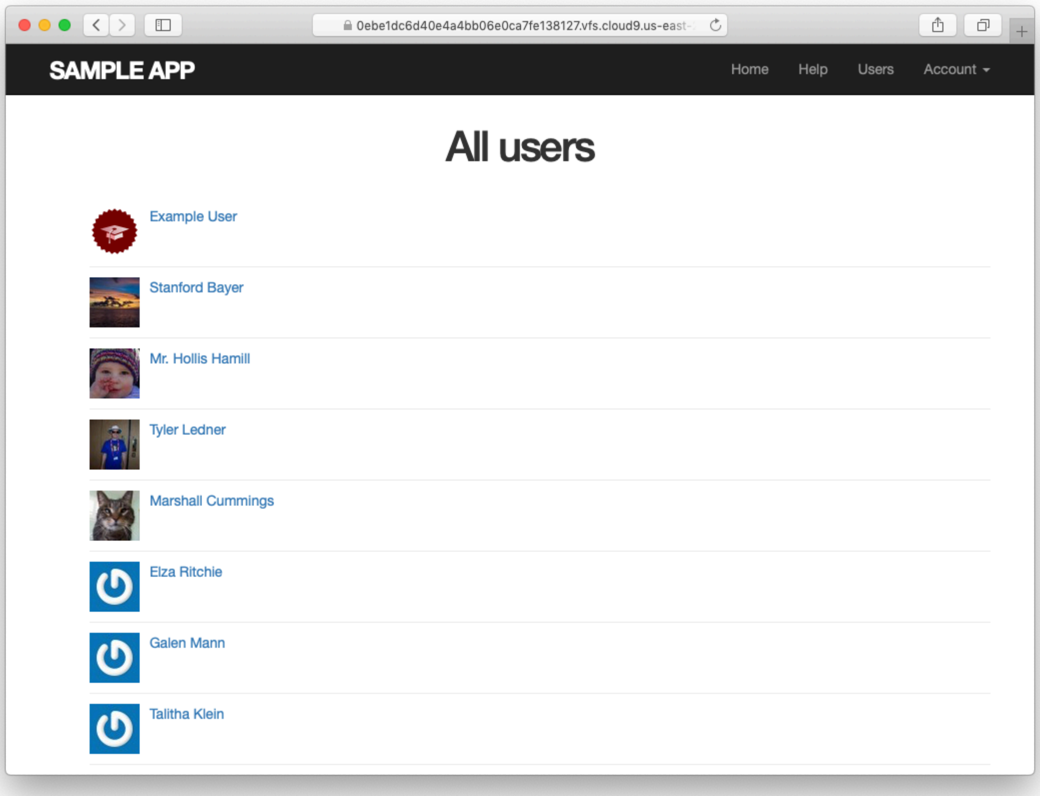


Figure 10.10: The users index page with 100 sample users.

After running the **`db:seed`** Rake task, our application should have 100 sample users. As seen in Figure 10.10, I've taken the liberty of associating the first few sample addresses with Gravatars so that they're not all the default Gravatar image.

### 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 trying to visit the edit page of another user results in a redirect as required by Section 10.2.2.

### 10.3.3 Pagination

Our original user doesn't suffer from loneliness any more, but now we have the opposite problem: our user has *too many* companions, and they all appear on the same page. Right now there are a hundred, which is already a reasonably large number, and on a real site it could be thousands. The solution is to *paginate* the users, so that (for example) only 30 show up on a page at any one time.

There are several pagination methods available in Rails; we'll use one of the simplest and most robust, called will_paginate. To use it, we need to include both the `will_paginate` gem and `bootstrap-will_paginate`, which configures will_paginate to use Bootstrap's pagination styles. The updated **`Gemfile`** appears in Listing 10.44.[12]

**Listing 10.44:** Including `will_paginate` in the **`Gemfile`**.

```
source 'https://rubygems.org'

gem 'rails',                   '6.0.1'
gem 'bcrypt',                  '3.1.13'
gem 'faker',                   '2.1.2'
gem 'will_paginate',           '3.1.8'
gem 'bootstrap-will_paginate', '1.0.0'
.
.
.
```

Then run **`bundle install`**:

```
$ bundle install
```

You should also restart the webserver to ensure that the new gems are loaded properly.

[12]As always, you should use the version numbers listed at gemfiles-6th-ed.railstutorial.org instead of the ones listed here.

To get pagination working, we need to add some code to the index view telling Rails to paginate the users, and we need to replace **`User.all`** in the **`index`** action with an object that knows about pagination. We'll start by adding the special **`will_paginate`** method in the view (Listing 10.45); we'll see in a moment why the code appears both above and below the user list.

**Listing 10.45:** The users index with pagination.
*`app/views/users/index.html.erb`*

```
<% provide(:title, 'All users') %>
<h1>All users</h1>

<%= will_paginate %>

<ul class="users">
  <% @users.each do |user| %>
    <li>
      <%= gravatar_for user, size: 50 %>
      <%= link_to user.name, user %>
    </li>
  <% end %>
</ul>

<%= will_paginate %>
```

The **`will_paginate`** method is a little magical; inside a **`users`** view, it automatically looks for an **`@users`** object, and then displays pagination links to access other pages. The view in Listing 10.45 doesn't work yet, though, because currently **`@users`** contains the results of **`User.all`** (Listing 10.36), whereas **`will_paginate`** requires that we paginate the results explicitly using the **`paginate`** method:

```
$ rails console
>> User.paginate(page: 1)
  User Load (1.5ms)  SELECT "users".* FROM "users" LIMIT 11 OFFSET 0
   (1.7ms)  SELECT COUNT(*) FROM "users"
=> #<ActiveRecord::Relation [#<User id: 1,...
>> User.paginate(page: 1).length
  User Load (3.0ms)  SELECT "users".* FROM "users" LIMIT ? OFFSET ?  [["LIMIT", 30],
  ["OFFSET", 0]]
=> 30
```

Note that **`paginate`** takes a hash argument with key **`:page`** and value equal to the page requested. **`User.paginate`** pulls the users out of the database one chunk at a time (30 by default), based on the **`:page`** parameter. So, for example, page 1 is users 1–30, page 2 is users 31–60, etc. If **`page`** is **`nil`**, **`paginate`** simply returns the first page. (The console result above shows 11 results rather than 30 due to a console limit in Active Record itself, but calling the **`length`** method bypasses this restriction.)

Using the **`paginate`** method, we can paginate the users in the sample application by using **`paginate`** in place of **`all`** in the **`index`** action (Listing 10.46). Here the **`page`** parameter comes from **`params[:page]`**, which is generated automatically by **`will_paginate`**.

**Listing 10.46:** Paginating the users in the **`index`** action.
*`app/controllers/users_controller.rb`*

```
class UsersController < ApplicationController
  before_action :logged_in_user, only: [:index, :edit, :update]
  .
  .
  .
  def index
    @users = User.paginate(page: params[:page])
  end
  .
  .
  .
end
```

The users index page should now be working, appearing as in Figure 10.11. (On some systems, you may have to restart the Rails server at this point.) Because we included **`will_paginate`** both above and below the user list, the pagination links appear in both places.

If you now click on either the 2 link or Next link, you'll get the second page of results, as shown in Figure 10.12.

## Exercises

Solutions to the exercises are available to all Rails Tutorial purchasers here.

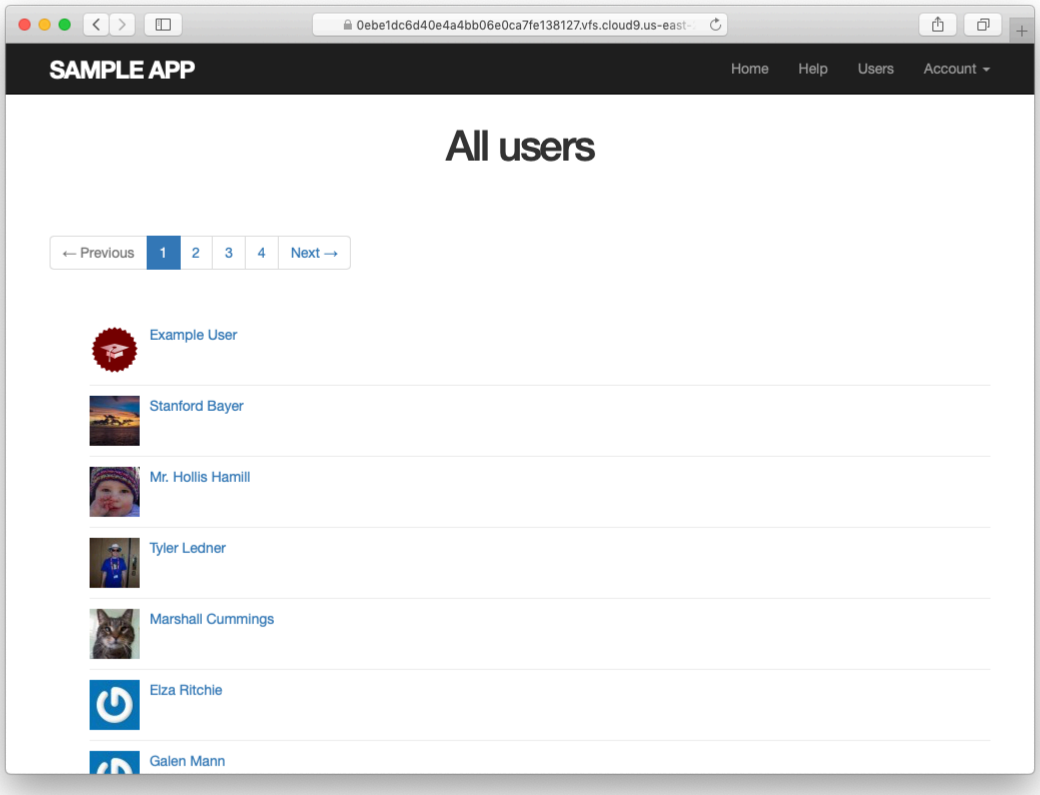


Figure 10.11: The users index page with pagination.

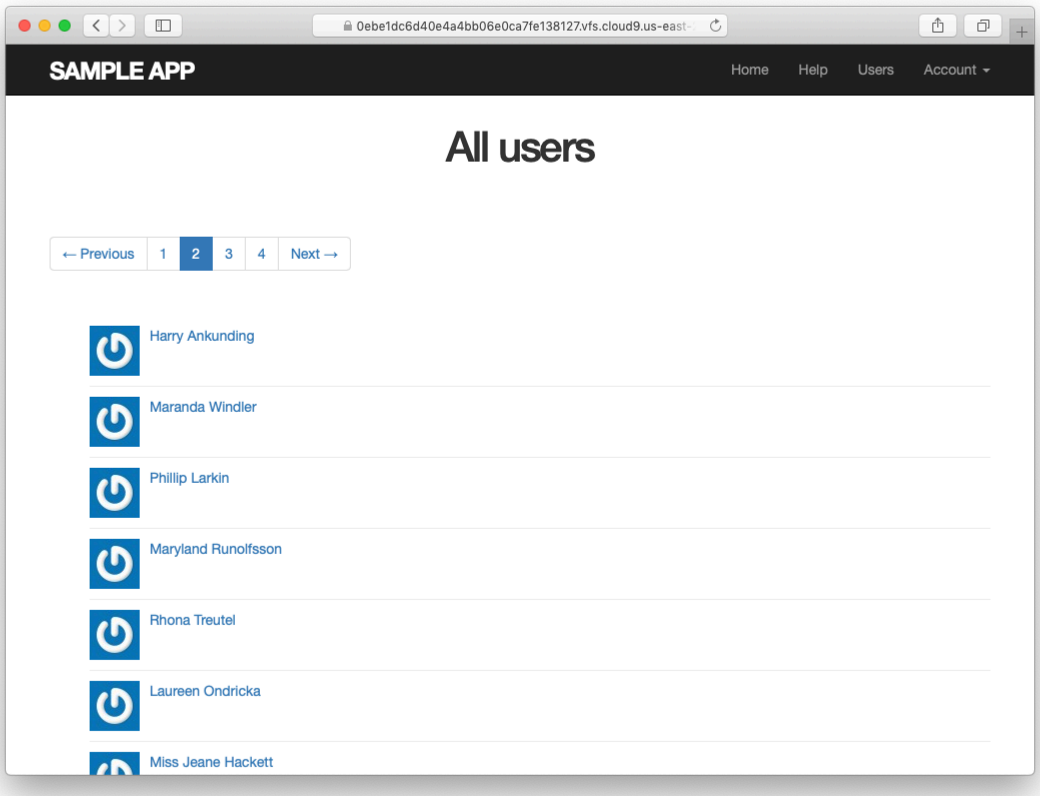


Figure 10.12: Page 2 of the users index.

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 at the console that setting the page to **`nil`** pulls out the first page of users.
2. What is the Ruby class of the pagination object? How does it compare to the class of **`User.all`**?

### 10.3.4 Users index test

Now that our users index page is working, we'll write a lightweight test for it, including a minimal test for the pagination from Section 10.3.3. The idea is to log in, visit the index path, verify the first page of users is present, and then confirm that pagination is present on the page. For these last two steps to work, we need to have enough users in the test database to invoke pagination, i.e., more than 30.

We created a second user in the fixtures in Listing 10.23, but 30 or so more users is a lot to create by hand. Luckily, as we've seen with the user fixture's **`password_digest`** attribute, fixture files support embedded Ruby, which
means we can create 30 additional users as shown in Listing 10.47. (Listing 10.47 also creates a couple of other named users for future reference.)

**Listing 10.47:** Adding 30 extra users to the fixture.
*`test/fixtures/users.yml`*

```
michael:
  name: Michael Example
  email: michael@example.com
  password_digest: <%= User.digest('password') %>

archer:
  name: Sterling Archer
  email: duchess@example.gov
  password_digest: <%= User.digest('password') %>

lana:
  name: Lana Kane
  email: hands@example.gov
  password_digest: <%= User.digest('password') %>

malory:
  name: Malory Archer
  email: boss@example.gov
  password_digest: <%= User.digest('password') %>

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

With the fixtures defined in Listing 10.47, we're ready to write a test of the users index. First we generate the relevant test:

```
$ rails generate integration_test users_index
      invoke  test_unit
      create    test/integration/users_index_test.rb
```

The test itself involves checking for a **`div`** with the required **`pagination`** class and verifying that the first page of users is present. The result appears in Listing 10.48.

**Listing 10.48:** A test of the users index, including pagination. GREEN
*`test/integration/users_index_test.rb`*

```
require 'test_helper'

class UsersIndexTest < ActionDispatch::IntegrationTest

  def setup
    @user = users(:michael)
  end

  test "index including pagination" do
    log_in_as(@user)
    get users_path
    assert_template 'users/index'
    assert_select 'div.pagination'
    User.paginate(page: 1).each do |user|
      assert_select 'a[href=?]', user_path(user), text: user.name
    end
  end
end
```

The result should be a GREEN test suite:

**Listing 10.49:** 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. By commenting out the pagination links in Listing 10.45, confirm that the test in Listing 10.48 goes RED.
2. Confirm that commenting out only *one* of the calls to will_paginate leaves the tests GREEN. How would you test for the presence of both sets of will_paginate links? *Hint*: Use a count from Table 5.2.

## 10.3.5 Partial refactoring

The paginated users index is now complete, but there's one improvement I can't resist including: Rails has some incredibly slick tools for making compact views, and in this section we'll refactor the index page to use them. Because our code is well-tested, we can refactor with confidence, assured that we are unlikely to break our site's functionality.

The first step in our refactoring is to replace the user `li` from Listing 10.45 with a `render` call (Listing 10.50).

**Listing 10.50:** The first refactoring attempt in the index view. RED
*app/views/users/index.html.erb*

```
<% provide(:title, 'All users') %>
<h1>All users</h1>

<%= will_paginate %>

<ul class="users">
  <% @users.each do |user| %>
    <%= render user %>
  <% end %>
</ul>

<%= will_paginate %>
```

Here we call **`render`** not on a string with the name of a partial, but rather on a **`user`** variable of class **`User`**;[13] in this context, Rails automatically looks for a partial called **`_user.html.erb`**, which we must create (Listing 10.51).

**Listing 10.51:** A partial to render a single user. GREEN
*app/views/users/_user.html.erb*

```
<li>
  <%= gravatar_for user, size: 50 %>
  <%= link_to user.name, user %>
</li>
```

This is a definite improvement, but we can do even better: we can call **`render`** *directly* on the **`@users`** variable (Listing 10.52).

**Listing 10.52:** The fully refactored users index. GREEN
*app/views/users/index.html.erb*

```
<% provide(:title, 'All users') %>
<h1>All users</h1>
```

[13]The name **`user`** is immaterial—we could have written **`@users.each do |foobar|`** and then used **`render foobar`**. The key is the *class* of the object—in this case, **`User`**.

```
<%= will_paginate %>

<ul class="users">
  <%= render @users %>
</ul>

<%= will_paginate %>
```

Here Rails infers that **`@users`** is a list of **`User`** objects; moreover, when called with a collection of users, Rails automatically iterates through them and renders each one with the **`_user.html.erb`** partial (inferring the name of the partial from the name of the class). The result is the impressively compact code in Listing 10.52.

As with any refactoring, you should verify that the test suite is still GREEN after changing the application code:

**Listing 10.53:** 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. Comment out the **`render`** line in Listing 10.52 and confirm that the resulting tests are RED.

## 10.4 Deleting users

Now that the users index is complete, there's only one canonical REST action left: **`destroy`**. In this section, we'll add links to delete users, as mocked up in Figure 10.13, and define the **`destroy`** action necessary to accomplish the