CS 416
Web Programming

Ruby on RAILS
Chapter 10

Dr. Williams
Central Connecticut State University

A look ahead

- First look session management & authentication

```
git checkout -b updating-users
```

Editing data - the user

- Existing routes – rails routes

```
edit_user GET    /users/:id/edit(.:format) users#edit
     user GET    /users/:id(.:format)      users#show
          PATCH  /users/:id(.:format)      users#update
          PUT    /users/:id(.:format)      users#update
```

- Pass existing model record to view

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

Edit form

```
<% provide(:title, "Edit user") %>
<h1>Update your profile</h1>
<div class="row">
  <div class="col-md-6 col-md-offset-3">
    <%= form_for(@user) do |f| %>
      <%= render 'shared/error_messages' %>

      <%= f.label :name %>
      <%= f.text_field :name, class: 'form-control' %>

      <%= f.label :email %>
      <%= f.email_field :email, class: 'form-control' %>

      <%= 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 "Save changes", class: "btn btn-primary" %>
    <% end %>
    <div class="gravatar_edit">
      <%= gravatar_for @user %>
      <a href="http://gravatar.com/emails" target="_blank" rel="noopener">change</a>
    </div>
  </div>
</div>
```

Security note - protecting external links
rel="noopener"

- Link to Gravatar site

```
<a href="http://gravatar.com/emails" target="_blank"
        rel="noopener">change</a>
```

- By default when open in new target, target window has access to javascript opener for origin window
  - Basically means new tab can open new content in the original tab which can be a problem in terms of phishing

```
<script>
  opener.location = 'https://fakeamazon.com/login';
</script>
```

- See https://mathiasbynens.github.io/rel-noopener/ for example

Resulting form

- View will automatically populate any field where name matches object form is for

```
<%= form_for(@user) do |f| %>
  <%= f.label :name %>
      <%= f.text_field :name, class: 'form-control' %>
  ...
<% end %>
```

- Recall path for update:

```
 user PATCH  /users/:id(.:format)      users#update
```

- Generated update form – map to REST operation:

```
<form id="edit_user_1" action="/users/1" method="post">
  <input type="hidden" name="_method" value="patch" />
```

Updating paths in header

- Use defined RESTful paths and session info from last time

```
<%= link_to "Profile", current_user %>
<%= link_to "Settings", edit_user_path(current_user) %>
```

Extract form commonality to partial

- Form data is same for new and edit **except button text** so move to partial:
  views/users/_form.html.erb

```
<%= f.submit yield(:button_text), class: "btn btn-primary" %>
```

- Pass specific button text for view

```
<% provide(:button_text, "Save changes") %>

<% provide(:button_text, "Create account") %>
```

- Render form

```
<%= render 'form' %>
```

Handling update submission

- Read user input protected with strong parameters, redirect if validation fails

```
def update
    @user = User.find(params[:id])
    if @user.update_attributes(user_params)
      # Handle a successful update.
      flash[:success] = "Profile updated"
      redirect_to @user
    else
      render 'edit'
    end
 end
```

Allowing null password on update

- has_secure_password ensures a password_digest specified in model, allowing to be null results in not being required on update, but still required on creation

```
class User < ApplicationRecord
  …
  has_secure_password
  validates :password, presence: true,
    length: { minimum: 6 },
    allow_nil: true
  …
end
```

Authentication/Authorization

- Covered authenticating user – verifying user knew password of user they said they were

- Authorization of site
  - Verify only authorized users can access parts of site
    - Currently if know url can view any page even if not on menu
    - Desired – might be allowed -> prompt to login
  - Verify user is allowed to do what they are trying to do
    - Change so can only modify logged on user
    - Desired – never allowed -> redirect to home

Add tests for security

- Add test *fixture* data – User for testing
  - Need helper method for password digest for User

```
 # Returns the hash digest of the given string.
def User.digest(string)
  cost = ActiveModel::SecurePassword.min_cost ? BCrypt::Engine::MIN_COST :
                                                BCrypt::Engine.cost
  BCrypt::Password.create(string, cost: cost)
end
```

  - Add a test user in *test/fixtures/users.yml*

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

Add helpers for session security

- Add to *test/test_helper.rb*

```
class ActiveSupport::TestCase
  fixtures :all

  # Log in as a particular user.
  def log_in_as(user)
    session[:user_id] = user.id
  end
end

class ActionDispatch::IntegrationTest

  # Log in as a particular user.
  def log_in_as(user, password: 'password')
    post login_path, params: { session: { email: user.email,
                                          password: password}}
  end
end
```

Add tests for not logged in

- Add check that pages that should be protected are:

```
class UsersControllerTest < ActionDispatch::IntegrationTest
  def setup
    @user = users(:michael)
  end
  ...
  test "should redirect edit when not logged in" do
    get edit_user_path(@user)
    assert_not flash.empty?
    assert_redirected_to login_url
  end

  test "should redirect update when not logged in" do
    patch user_path(@user), params: { user: { name: @user.name,
                                              email: @user.email } }
    assert_not flash.empty?
    assert_redirected_to login_url
  end
end
```

`rails test` (as wanted)

Protecting through filters

- Rails approach, allow controller methods to be protected through filters before being processed

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

  ...

  private
    ...

    # Before filters

    # Confirms a logged-in user.
    def logged_in_user
      unless logged_in?
        flash[:danger] = "Please log in."
        redirect_to login_url
      end
    end
```

```
rails test
```

Add tests for not correct user

- Add check can only edit logged in user:
  - Add second user to fixture file
  - Add tests

```
class UsersControllerTest < ActionDispatch::IntegrationTest
  def setup
    @user = users(:michael)
    @other_user = users(:chad)
  end
  ...
 test "should redirect edit when logged in as wrong user" do
    log_in_as(@other_user)
    get edit_user_path(@user)
    assert flash.empty?
    assert_redirected_to root_url
  end

  test "should redirect update when logged in as wrong user" do
    log_in_as(@other_user)
    patch user_path(@user), params: { user: { name: @user.name,
                                              email: @user.email } }
    assert flash.empty?
    assert_redirected_to root_url
  end
end
```

rails test (as wanted)

More protecting through filters

- Add filter to confirm same user

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

  # Before filters

  # Confirms the correct user.
  def correct_user
    @user = User.find(params[:id])
    unless @user == current_user
      flash[:danger] = "You are not authorized to do that."
      redirect_to(root_url)
    end
  end
```

rails test

Expand protection

- Require being logged in to see index of users and only user can only see details of themselves

```
before_action :logged_in_user, only: [:index,:show,:edit, :update]
before_action :correct_user, only: [:show,:edit, :update]
```

Friendly forwarding

- Currently when try to access protected page, prompts user to login then sends them to their profile page
- Desired – redirect them to login, but then after login forward to **original destination**
- Logical tasks
  - If redirecting first store original destination
  - If logging in see if there was an original destination

Session helper functions

- In *app/helpers/sessions_helper.rb*
  - Helper function to store original destination

```
# Stores the URL trying to be accessed.
def store_location
  session[:forwarding_url] = request.original_url if request.get?
end
```

- Helper function to redirect to original if present otherwise default destination

```
# Redirects to stored location (or to the default).
def redirect_back_or(default)
  redirect_to(session[:forwarding_url] || default)
  session.delete(:forwarding_url)
end
```

Storing the location

- In *app/controllers/users_controller.rb* filter

```
# Confirms a logged-in user.
def logged_in_user
  unless logged_in?
    store_location
    flash[:danger] = "Please log in."
    redirect_to login_url
  end
end
```

Using the location on login

- In app/controllers/sessions_controller.rb

```
def create
   user = User.find_by(email: params[:session][:email].downcase)
   if user && user.authenticate(params[:session][:password])
     # Log the user in and redirect to the user's show page.
     log_in user
     #redirect_to user
     redirect_back_or user
   else
     flash.now[:danger] = 'Invalid email/password combination'
     render 'new'
   end
 end
```

Faking sample data

- Generating sample data can be time consuming, faker gem can help

```
gem 'faker',          '1.6.6'
```

- Then seed data using *db/seeds.rb* then **`rails db:seed`**

```
User.create!(name:  "Example User",
             email: "example@railstutorial.org",
             password:              "foobar",
             password_confirmation: "foobar")

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
```

Adding pagination on index and search

- Add gems

```
gem 'will_paginate',              '3.1.0'
gem 'bootstrap-will_paginate', '0.0.10'
```

- Add will_paginate to add page navigation around results (when necessary)

```
<%= will_paginate %>
...
<%= will_paginate %>
```

- To work with Ajax results add `method: "get"` in form tag

Modify query in controller for paging

(Also clean up and shift search to index, make index js version too)

```
def index
  if params && params[:search]
    name = params[:search] + '%'
    @users = User.where(['name LIKE ?', name]
       ).paginate(:page => params[:page]).order('id DESC')
  else
    @users = User.paginate(page: params[:page])
  end

  respond_to do |format|
    format.html
    format.js
  end
end
```