CS 416
Web Programming

Ruby on RAILS
Chapter 8-9 User login, Ajax search

Dr. Williams
Central Connecticut State University

A look ahead

- First look session management & authentication

- Adding Ajax search of users

Create sessions controller

- Generate Sessions controller and *new* view

```
rails generate controller Sessions
new
```

- Add RESTful routes for sessions

```
get '/login',          to: 'sessions#new'
post '/login',         to: 'sessions#create'
delete '/logout',      to: 'sessions#destroy'
```

Modify header partial for login_path

Login form

```
<% provide(:title, "Log in") %>
<h1>Log in</h1>

<div class="row">
  <div class="col-md-6 col-md-offset-3">
    <%= form_for(:session, url: login_path) do |f| %>

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

      <%= f.label :password %>
      <%= f.password_field :password, class: 'form-control' %>

      <%= f.submit "Log in", class: "btn btn-primary" %>
    <% end %>

    <p>New user? <%= link_to "Sign up now!", signup_path %></p>
  </div>
</div>
```

Finding and authenticating user

```
def new
 end

 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.
   else
     flash.now[:danger] = 'Invalid email/password combination'
     render 'new'
   end
 end

 def destroy
 end
```

Including session functionality across the site

- Modify Application controller to include Sessions Helper

```
class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception
  include SessionsHelper
end
```

- Create common log in

```
module SessionsHelper

  # Logs in the given user.
  def log_in(user)
    session[:user_id] = user.id
  end
end
```

Completing login

```
if user && user.authenticate(params[:session][:password])
  log_in user
  redirect_to user
else
  ...
```

Completing sessions helper – get current user and check if logged in

```
# Returns the current logged-in user (if any).
def current_user
  @current_user ||= User.find_by(id: session[:user_id])
end

# Returns true if the user is logged in, false otherwise.
def logged_in?
  !current_user.nil?
end
```

Creating dynamic header

```
<% 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", '#' %></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 %>
```

Add javascript libraries

- To the application.js file add
  - Bootstrap for drop down menu

app/assets/javascripts/application.js

```
//= require jquery
//= require bootstrap
//= require jquery_ujs
//= require turbolinks
//= require_tree .
```

Login upon signup

- Add call to login helper method in create

```
if @user.save
  log_in @user
  flash[:success] = "Welcome to the Sample App!"
  redirect_to @user
else
  render 'new'
end
```

Adding logout

- Add helper method

```
# Logs out the current user.
def log_out
  session.delete(:user_id)
  @current_user = nil
end
```

- Use in session controller

```
def destroy
  log_out
  redirect_to root_url
end
```

Adding search

- Existing RESTful Route to list users:

```
users GET    /users(.:format)    users#index
```

- Add the functionality so the page lists all users, but filters down users by Ajax

- Start by completing index path
  - Add index method to controller

```
def index
  @users = User.all
end
```

Create views

- Basic layout is placeholder for table with results to be populated
- Simple index view with partial for results

```
<h1>User search</h1>
<div id="results">
  <%= render 'results' %>
</div>
```

Results partial

- In *_results.html.erb*

```
<table width="100%" border="1">
  <thead>
    <tr>
      <th width="25%">Name</th>
      <th width="50%">Email</th>
      <th width="25%"></th>
    </tr>
  </thead>

  <tbody>
    <%= render @users %>
  </tbody>
</table>
```

Indicates use default partial for user

User partial

- In *_user.html.erb*

```
<tr>
    <td><%= user.name %></td>
    <td><%= user.email %></td>
    <td><%= link_to 'Show', user %></td>
</tr>
```

- Now if specify to show a user in a page and don't explicitly specify format this will be used as default display

Adding search

- Add route for where to post our search query

```
post '/search',  to: 'users#search'
```

- Add collection of search terms to index page:

```
<%= form_tag(search_path, id: "search_form") do %>
  <%= text_field_tag :search, params[:search] %>
  <%= submit_tag "Search", name: nil %>
<% end %>
```

Add search path

- Controller method

```
def search
  name = params[:search] + '%'
  @users = User.where(['name LIKE ?', name])
end
```

- Add simple view search.html.erb

```
<div id="results">
  <%= render 'results' %>
</div>
```

Now working form post search results…

Making search Ajax

- Modify index to make the form *remote*

```
<%= form_tag(search_path, remote: true, id:
"search_form") do %>
```

- Now add to controller how to respond to each format:

```
def search
    name = params[:search] + '%'
    @users = User.where(['name LIKE ?', name])
    respond_to do |format|
      format.html
      format.js
    end
  end
```

Ajax response

- Format of response says which view to render, JS response says render ***search.js.erb*** rather than ***search.html.erb***

- In *search.js.erb* have Jquery insert output of results partial into results div tag

```
$("#results").html(
"<%= escape_javascript(render("results")) %>");
```

Presto! Ajax results

Results as you type

- To make the results filter as we type rather than submit add JS to all search pages..

- In application.js

```
$(document).on('turbolinks:load',function() {
    $('#search').on('keyup', function() {
        $('#search_form').submit();
    });
});
```

Authentication/authorization

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

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
```

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
```

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
```

Adding update of 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>
</div>
```

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
```