INFO 2310

Topics in Web Programming
Ruby on Rails

Last week on INFO2310

- Added our first ActiveRecord relation
- Discussed the application.css file
- Used ActiveRecord callbacks and a conditional validator to add a password field to our User Model
- You wrote the User.authenticate function

My Solution*

```
def has_password?(raw_password)
  hashed_password == encrypt(raw_password)
end

def self.authenticate(email, plain_text_password)
  user = User.find_by_email(email)
  user && user.has_password?(plain_text_password) ? user : nil
end

# Ternary operator, as seen in PHP and Java
# (boolean ? value_if_true : value_if_false)
```

Also Professor Williamson's from the previous offering and Michael Hartl's from the online tutorial...

Week 5

Signing in/out
&& Customizations

Go ahead and login to Amazon and start your EC2 instances

Hold the phone!!

You told us using a web framework would save us from rewriting components that are common to many websites...

So why are we writing a *login* system?

has_secure_password
http://api.rubyonrails.org/classes/ActiveModel/SecurePassword/ClassMethods.html

Most popular existing solution (from what I can tell):
* https://github.com/plataformatec/devise

Login to PuTTY

- open PuTTY
- On the left panel, navigate to Connection->SSH->Auth
  - Next to "Private key file for authentication:", click "Browse", and select the *.ppk file you created on the previous step.
- Then, navigate to Connection->Data
  - For "Auto-login username", type "ec2-user"
- Navigate to "Session" (the very top)
  - Copy the "Public Domain Name" of your EC2 instance; you can see this on the instances page of the Amazon console, when an instance is selected
  - Paste it into the "Host Name (or IP address)" field
- Click "Open"

Login to WinSCP

- Open WinSCP
  - Paste in your domain to "Host name", as you did in PuTTY
  - Type "ec2-user" for the "User name"
  - Click "..." to select your private key file
  - Click "Login"
- Set NotePad++ as the default editor.
  - Click Options->Preferences
  - Select "Editors" from the left tab
  - Click Add
  - Select "External Editor"
  - Find NotePad++ (C:\Program Files (x86)\Notepad++)
  - Click "Open", then "Okay"
  - Drag it to the top of the editor list

Today's branch

Since we are working on a new feature today, let's start on a feature branch

```
git status                  # should display nothing to commit
git checkout -b login       # checkout a new branch
git branch                  # view branches
```

Specs

We will spend today working through the **spec/requests/session_spec.rb** file, so let's copy that over from lecture_5.txt.

As we progress today you will get more and more of the specs in this file to pass.

With luck, by the end of today, all of them will be passing and you will have implemented a working login system!

Remembering Logins

Now we have a function to checks a user's credentials

=> User.authenticate

What do we do with it?

We want a user to login, and then begin a session, which persists until they log out.

Our Strategy

- Show users a login form
- When they submit the form
  - If their information is accurate, save their user_id in a cookie.
  - Subsequent requests we will determine the authenticated user by looking up that user_id in our database

Let's be RESTful

While we won't actually be storing any session data in the database (we'll use cookies), we will model logins in a RESTful manner just the same.

We will think of logging in as *creating* a Session *resource*. Logging out is *destroying* that *resource*.

Remember?

config/routes.rb

resources :users

HTTP Verb	Path	Action	Helper function
GET	/users	index	users_path
GET	/users/new	new	new_user_path
POST	/users	create	users_path
GET	/users/:id	show	user_path(user)
GET	/users/:id/edit	edit	edit_user_path(user)
PUT	/users/:id	update	user_path(user)
DELETE	/users/:id	destroy	user_path(user)

We don't quite need all that

config/routes.rb

resources :sessions, only: [:new, :create, :destroy]

HTTP Verb	**Path**	**Action**	**Helper function**
GET	/sessions/new	new	new_session_path
POST	/sessions	create	sessions_path
DELETE	/sessions/:id	destroy	session_path(user)

Along with the mappings from URL to controller and function, this gives us some helper functions to access these URLs in other parts of our app.

```
new_session_path        => "/sessions/new"
sessions_path           => "/sessions"
session_path(id: 5)     => "/sessions/5"
```

And the controller

app/controllers/sessions_controller.rb

```
class SessionsController < ApplicationController

  def new
  end

  def create
  end

  def destroy
  end

end
```

Login Form Step 1

1. Create a new file at **app/views/sessions/new.html.erb**
   a. You can use the template in lecture_5.txt to start

2. Change the "Logout" link in the header to be a "Login" link which points to your new login form. You can edit this in **app/views/layouts/_header.html.erb**. After you do this, you should be able to click "Login" and view the file you just created.

Here's how you can run just the specs relevant to this task:

bundle exec rspec spec/requests/session_spec.rb -e "login form link should be displayed in the header"

Once this is passing, you're ready to move on.

A login form

the **form_for** helper is used to produce the HTML for a form.

We've seen it before in

app/views/users/_form.html.erb

app/views/micro_posts/_form.html.erb

form_for produces the HTML for a web form. If you pass it an instance of an ActiveRecord class, as we did in the two partials above, it will infer the appropriate URL and action (POST/PUT) for the form.

However, unlike before, we don't have a Session ActiveRecord model. We can still use form_for, we just have to pass it more information; specifically, the model the form is for, and the url it should submit to.

form_for

```
<%= form_for(:document, url: documents_path) do |f| %>

  <div class="field">
    <%= f.label :title %><br />
    <%= f.text_field :title %>
  </div>
  <div class="field">
    <%= f.label :content %><br />
    <%= f.text_area :content %>
  </div>
  <div class="actions">
    <%= f.submit "Create Document" %>
  </div>

<% end %>
```

Login Form Step 2

1. Fill in **app/views/sessions/new.html.erb** with the code needed to create an HTML form.
   a. Set the resource and correct path
   b. add a text_field for email
   c. add a password_field for password
   d. a button to "Login"

2. The form should POST to "/sessions", but don't worry about it working yet. In fact, it's fine if it crashes when you submit it.

Here's how you can run just the specs relevant to this task:

bundle exec rspec spec/requests/session_spec.rb -e "login form elements"

Once this is passing, you're ready to move on.

Flash Variables

Are great ways to display notification or error messages.

Rails persists them to the subsequent request and then deletes them.

Here's one in action

```
def set_user_name_to_matt
    user = User.find_by_id(params[:id])

    if user && user.update_attributes(name: "matt")
        flash[:notice] = "Updated name to Matt"
        redirect_to user_path(user)
    else
        flash[:error] = "Unable to update name to Matt"
        redirect_to root_path
    end
end
```

Adding flash variables to our layout

app/views/layouts/application.html.erb

```
<div class="container">

  <% flash.each do |key, value| %>
    <div class="alert alert-<%= key %>"><%= value %></div>
  <% end %>

  <%= yield %>
  <hr />
  <%= render 'layouts/footer' %>
</div>
```

See full file in lecture_5.txt

SessionsController

Fill in the code for the **create** function in **app/controllers/session_controller.rb**

Use your **User.authenticate** function that you wrote for HW

1. Inside the controller function you can use
   a. **params[:session][:email]** and
   b. **params[:session][:password]** to get the form inputs.
2. If **User.authenticate** returns a user
   a. set a flash :notice of "Welcome, <their email address>!"
   b. redirect to the users profile page
      i. if you have a user object, **user_path(user)** will be the path to their profile page
3. If **User.authenticate** returns **nil**
   a. set a flash :error of "Invalid email/password combination"
   b. redirect back to the login form

Here's how you can run just the specs relevant to this task:

bundle exec rspec spec/requests/session_spec.rb -e "login form submission"

Once this is passing, you're ready to move on. We still aren't *actually* logging them in yet; that's next.

Cookies!

Finally, we are at a spot to actually log people in. Let's use a cookie to remember who the current user is.

In Rails, we can write cookies by setting keys on the **cookies** hash.

```
cookies[:current_user_id] = user.id
```

Then in subsequent requests we can read them back:

```
current_user_id = cookies[:current_user_id]
```

Logging in

Let's remember the current user id on successful logins using a cookie*.

Next we need some helper methods to facilitate working with the currently authenticated user.

We'll have several, so lets start a new module, at **app/helpers/session_helper.rb**

*this is silly and horrifically insecure; I could become any user I want by just sending a cookie with their user_id.

What's a module?

A set of functions with a common purpose that can be included in other Ruby classes.

Unlike inheritance, a Ruby class may include many modules (but may only inherit from one parent class).

Sometimes they are called "mixins" because they "mixin" functionality to a class. This is a very powerful tool for reusing code.

app/helpers/session_helper.rb

```
module SessionHelper

  def sign_in(user)
    cookies[:current_user_id] = user.id
    self.current_user = user
  end

  def sign_out_user
    cookies.delete :current_user_id
  end

  def current_user=(user)
    @current_user = user
  end

  def current_user
    @current_user ||= User.find_by_id(cookies[:current_user_id])
  end

  def signed_in?
    !current_user.nil?
  end
end
```

Helpers

Helpers are automatically included for us to use in views, but we will need to include our helper ourselves if we wish to use the functions in controllers. Which we do.

app/controllers/application_controller.rb

```
class ApplicationController < ActionController::Base
  protect_from_forgery
  include SessionHelper
end
```

Including the SessionHelper module is effectively the same as defining the methods *on* the ApplicationController class; but it's much cleaner to keep them in a separate module

Customize our header

- Add a call to **sign_in(user)** to your **create** function, after a user successfully authenticates
- Using the **signed_in?** and **current_user** helpers, add logic to **app/views/layouts/_header.html.erb** such that if someone is logged in:
  - Display a "Logout" link instead of a "Login" link
    - for the logout link, you'll need to pass an additional "method: : delete" argument to the "link_to" function, as described here:
    - http://api.rubyonrails.org/classes/ActionView/Helpers/UrlHelper.html#method-i-link_to
  - Set the link for "My Profile" to point to the current_user's profile page
  - Set the link for "My Account" to point the page for editing the current_user
  - Hide the "Sign Up" button on the home page
- If someone is not logged in
  - Display a "Login" link instead of a "Logout" link
  - Hide the links for "My Profile" and "My Account"

When you're done, more specs should be passing:

bundle exec rspec spec/requests/session_spec.rb -e "customized

Logging out

In our sessions_controller, implement the **destroy** function.

The destroy function should

- logout the current user (we already have the function to do this in our **SessionHelper** module).
- set a flash notice on the page that reads "Logged out <email>"
- redirect to the home page.

Once these are passing, you got it
bundle exec rspec spec/requests/session_spec.rb -e "logging out"

Once those pass, all your tests should be passing :)
bundle exec rspec

Commit time

```
git status                          # see what we modified
git add -A                          # add all the changes
git commit -m "authentication"


git checkout master                 # merge it back into master
git merge login

git push origin master              # github
git push heroku master              # heroku
```

Today we...

- Created the login form
- Used a cookie to remember who the current user is
- Customized our layout for the current user
- Implemented logging out

OH
Now until 4pm
Monday from 7-8pm