Cybersecurity: Sign Up, In, and Out
CSC340

In this worksheet we'll continue our cybersecurity study for the website by making use of the authentication backend we wrote previously for the Microposter app.

I. Sign Up

1. As usual, please start with:
   ```
   cp -r microposter2 microposter3
   cd microposter3
   ```

2. We previously added password and password_confirmation as virtual attributes (not stored in the database). So we need to make use of these when a user is creating a new account.

3. Recall how scaffolding sets up creation of a new resource. The new action calls the new view, which loads the _form partial for the actual form, which generates a POST request to the create action. Review the code to remind yourself how this works.

4. So we need to edit the _form partial to include entry of the password and password_confirmation.
   a. Do this by adding to the content of the form_for body in the _form partial. Instead of f.text_field, use f.password_field so that we mask what the user types, as is expected on websites today.

5. When the user presses the button in the form, the create action is called. The first step in the create action is to obtain the parameters from the form to pass to the User constructor. Edit the user_params method so that the password and password_confirmation are also permitted, and thus returned by the user_params method.

6. Verify that you've made the above changes correctly, by creating a new user through the app, and then checking that it was created correctly in the rails console.

7. Note that since we made these changes in the _form partial, our changes will apply both for creating a new user, and for editing an existing user.

II. Sign In & Out

1. We need to be able to keep track of whether or not a user is logged in. Let's make a controller to manage this functionality. Make a controller called Sessions with just one action: new.

2. Manually add two more actions to the Sessions controller definition: create and destroy. We don't need views for these actions, though. That's why we didn't make them at the command line in the previous step.

3. In routes.rb, comment out or delete the `get 'sessions/new'` line. We want to create standard resource routes for new, create, and destroy, so that we can have easy access to them via named routes in other code. Do this with: `resources :sessions, only: [:new, :create, :destroy]`

    a. This is similar to the line that a scaffold would generate, except note that we're restricting the generated routes to just new, create, and destroy. We won't be implementing the other actions that normally get routed with the resources method (and scaffolding), so we're preventing routes from being generated for them.
    b. So now we have named routes for new_path, create_path, and destroy_path.

4. We're now ready to create routes for sign up, in, and out:
```
match '/signup', to: 'users#new', via: 'get'
match '/signin', to: 'sessions#new', via: 'get'
match '/signout', to: 'sessions#destroy', via: 'delete'
```

    a. Recall that this gives us signup_path, signin_path, and signout_path named routes.
    b. Note that we're using a DELETE request for signout, since we will be deleting the session created at sign-in.
    c. Of course, we could have done the signup route a long time ago - we've had users#new for a long time.

5. Create an appropriate form in the new view for the Sessions controller.
    a. We'll use form_for, of course, but instead of basing it on a model instance stored in an attribute, we'll pass information on the controller itself: `form_for(:session, url: sessions_path)`
    b. We might as well use some nice Bootstrap CSS here, so for the f.submit method call, include the parameter:
    `class: "btn btn-large btn-primary"`
    c. At the bottom, include a note saying "New user? Sign up now!" where "Sign up now!" is a link to the sign-up page.

6. We need to write the Sessions actions. For new, we actually won't need any code in the action. The new view will be called automatically, and that's all we need.

7. Recall that the create action is called when the form button in the new view is pressed. Use the following code, which we'll explore together:

```
def create
  @user = User.find_by(email: params[:session][:email].downcase)
  if @user && @user.authenticate(params[:session][:password])
    session[:user_id] = @user.id
    redirect_to @user
  else
    flash.now[:message] = 'Invalid email/password combination'
    render 'new'
  end
end
```

   a. In the first line, we get the email from the session, setting it all to lowercase before it is used to find the user.

   b. The if statement checks two things. First, does the user exist? This check will fail if the email provided doesn't belong to a registered user. Second, was the correct password entered for that user? This is checked using the authenticate method that we obtained by using the has_secure_password method in the User model.

   c. If the login is successful, then the user id is saved in the session, and we redirect to the details page for that user.

   d. If the login is unsuccessful, then a message is put in the flash and the new view is simply brought up again. Since render does not generate a new HTTP request, we need to use flash.now, rather than simply flash, to make the flash change occur for the current HTTP request instead of the next one.

   e. Of course to make this flash message show up in the new view, we need to add the following to the new view:
      ```
      <% if flash[:message] %>
      <%= flash[:message] %>
      <% end %>
      ```

8. We also need to write code for the destroy action. It's much easier than create. Just set the value stored in session[:user_id] to nil, and then call redirect_to on the home page.

9. Now fill in the remaining # address placeholders in application.html.erb.

10. Just to keep things simple for now, let's add one more link to the header navigation bar:
    ```
    <li><%= link_to "Sign out", signout_path, method: :delete %></li>
    ```

    a. Note the inclusion of the parameter to specify the HTTP request method. The default is GET, which is incorrect for signing out. Recall in routes.rb that we specified signing out as using a DELETE request, so we need to specify that here.

11. We can now sign in and out! Of course, this doesn't actually *do* anything for us yet... in the next worksheet we'll see how to limit access based on login.