Authentication/Authorization

INFO 2310:
Topics in Web Design and Programming

Want to get a job or internship in Information Science?

- Is your resume ready for companies to see, or would you like it to help you get that job?
- The ISSA will be having a Resume Workshop specifically tailored to Information Science with Craig Jones from Cornell Career Services
- Monday, October 6th, 2008 @ 4:30 in Upson 205
- We'll have food and prizes!

infoscience STUDENT ASSOCIATION

Announcement

Last class for INFO 2310 will be November 7, not October 31.

In our last episode…

- Last time, we defined two new models for users and comments.
- We also started connecting the models, and starting modifying the associated views to start working more like a blog.
- Still need to modify routes and controllers (and views) to be more blog-like.

Nested resources

- It really doesn't make sense for comments to be separate from posts; we shouldn't be able to create a separate comment #4 via a URL comments/4.
- We really want every comment connected to a post; the URLs for comments should be posts/2/comments/4.

We can do this by updating our routing table to declare that comments are a resource *nested* inside posts.

Get config/routes.rb. Remove the line

```
map.resources :comments
```

Modify the line

```
map.resources :posts
```

to

```
map.resources :posts,
  :has_many => :comments
```

We can see what the new routes are by typing '`rake routes`'.

We'd like someone to be able to add a comment to a post.

In app/views/posts/show.html.erb, we can add the line

```
<%= link_to "Add Comment",
  new_post_comment_url(@post) %> |
```

before the links to Edit and Back.

This breaks when we actually try to enter a new comment. A few things to fix:

- The view
- The comments controller

new.html.erb

Edit app/views/comments/new.html.erb to:

```
<h1>New comment</h1>
<% form_for :comment, :url => {:action => :create} do |f| %>
  <%= f.error_messages %>
  <p>
    <%= f.label :body %><br />
    <%= f.text_area :body %>
  </p>
  <p>
    <%= f.submit "Create" %>
  </p>
<% end %>

<%= link_to 'Back', post_url(@post) %>
```

Comments controller

For almost everything we want to do in the comments controller, we're going to want the post associated with the comment.

We can set up a call to make sure we get it each time.

before_filter

At the top of the comments controller (app/controllers/comments_controller.rb) we add

```
before_filter :grab_post
```

This gets called before each action. At the bottom we define this method

```
private
  def grab_post
    @post = Post.find(params[:post_id])
  end
```

Of course, there's an "after_filter" as well, if we ever needed it.

Comments controller

In fact, we then should redefine all the actions in the comments controller in (app/controllers/comments.rb).

See code snippets for the entire file.

```
class CommentsController < ApplicationController
 before_filter :grab_post

 def index
  @comments = Comment.find(:all)
 end

 def show
  @comment = Comment.find(params[:id])
end

 def new
  @comment = Comment.new
end

 def edit
  @comment = @post.comments.find(params[:id])
 end

 def create
   @comment =Comment.new(params[:comment])
  if (@post.comments << @comment)
     redirect_to post_url(@post)
  else
   render :action => :new
  end
end

def update
 @comment = @post.comments.find(params[:id])
  if @comment.update_attributes(params[:comment])
   redirect_to post_url(@post)
  else
   render :action => :edit
  end
end

def destroy
 comment = @post.comments.find(params[:id])
 @post.comments.delete(comment)
 redirect_to post_url(@post)
end

private
 def grab_post
  @post = Post.find(params[:post_id])
 end

end
```

To dos..

There's still plenty of work to do in fixing up the views and controllers to get them to do what you want…

But let's go on to our next topic…

Authentication/Authorization

We will want to be able to:

- Allow users to log in (and keep their password secure)
- Limit actions to users who have been authorized to do them (e.g. maybe only the author of a post can edit it).

We were bad

We created a user model with a password, but we did something bad. What was it?

Passwords in the clear

Right now our user password is in the database in cleartext. This is a *bad* idea.

```
create_table :users do |t|
  t.string :name
  t.string :email
  t.string :password

  t.timestamps
end
```

Let's change it so that we have a encrypted password and a salt in our DB; while we're at it, let's make sure the email string is not null.

Create a migration (`ruby script/generate migration crypt_password`).

```
class CryptPassword < ActiveRecord::Migration
  def self.up
    remove_column :users, :password
    add_column :users, :crypted_password, :string, :limit =>
    256
    add_column :users, :salt, :string
    change_column :users, :email, :string, :null => false
  end

  def self.down
    remove_column :users, :crypted_password
    remove_column :users, :salt
    add_column :users, :password, :string
  end
end
```

This is in the code snippets file.

Adding an accessor

Since 'password' isn't part of the User model given by the DB anymore, we need to allow for a way to get/set the unencrypted password for a User instance (which won't get saved to the DB).

In the User model (app/models/user), add the line

```
attr_accessor :password
```

(Remember from Lecture 1 on Ruby, this adds a standard getter/setter to an object).

Model callbacks

So what do we do with a password to make sure it is saved in encrypted form?

We can ask a model to call a function `before_validation`, `before_save`, or `after_create`.

In this case, we want the password to be encrypted before saving it.

Encryption

In app/models/users, add the line

```
before_save :encrypt_password, :unless =>
  lambda {|u| u.password.blank?}
```

This will call a function 'encrypt_password'.

What does the rest do? It passes in an 'anonymous function' that takes as input a user and checks if its password is blank, so we only do encryption on a save in case we are saving the plaintext password.

Doing the encryption

We're going to use a standard library, so at the very top of the User model (before the Class statement), add the line `require 'digest/sha2'`.

Then inside the model add

```
protected
  def encrypt_password
    self.salt = Digest::SHA256.hexdigest("--#{Time.now.to_s}-
  -#{email}--")
    self.crypted_password = encrypt(password)
    self.password = nil
  end

  def encrypt(password)
    Digest::SHA256.hexdigest("--#{salt}--#{password}--")
  end
```

Logs

A good thing

Lots of details of the various events occurring in your Rails application get stored in a log; open up blog/log/development.log and take a look around.

A bad thing

Search for the SQL `INSERT INTO "users"`.

What do you see?

Passwords in the clear

```
Processing UsersController#create (for 127.0.0.1 at 2008-09-24
   22:34:12) [POST]
  Session ID:
   BAh7BzoMY3NyZl9pZCIlNjg5MzEzOGQ2ZWE5YmI5YjJmNGYwMDU2ODg0NWZh
ZGUiCmZsYXNoSUM6JfdGlvbkNvbnRyb2xsZXI6OkZsYXNoOjpGbGFzaEhh
c2h7AAY6CkB1c2VkewA=--7670d43a2c040709b6ab697f171cf692498841ce
  Parameters: {"user"=>{"name"=>"David", "password"=>"MyPassword",
   "email"=>"dpw@cs.cornell.edu"}, "commit"=>"Create",
   "authenticity_token"=>"d92550adeea871720c84356fe3d26ffb54d62236",
   "action"=>"create", "controller"=>"users"}
   [4;36;1mUser Create (0.000000) [0m    [0;1mINSERT INTO "users"
   ("name", "updated_at", "password", "email", "created_at")
   VALUES('David', '2008-09-25 02:34:12', 'MyPassword',
   'dpw@cs.cornell.edu', '2008-09-25 02:34:12') [0m
Redirected to http://localhost:3000/users/1
Completed in 0.46800 (2 reqs/sec) | DB: 0.00000 (0%) | 302 Found
   [http://localhost/users]
```

Not good for our users' security…

A fix

We can fix this by adding a line to app/controllers/application.rb; whatever is in this file applies/is available to all controllers.

In this case we add

```
filter_parameter_logging :password
```

This removes from the log the value of any parameter hash whose key has a substring of 'password' in it.

More caution

What happens if someone does an HTTP PUT to /users/1 with params[:user][:salt] set to 'haxx0r!1'.

Yes, your salt gets reset and saved to the new value.

Some protection

In the user model, add the line

```
attr_accessible :email, :name,
:password
```

Then when a hash passed in to the user model, only these attributes are set; e.g.

```
User.new(params[:user])
```

only sets the :email, :name, :password from the params[:user] hash.

Logging in

Logging in

We could create logging in/out actions in our Users Controller, but instead we take the perspective that

- Logging in = Creating a new session
- Logging out = Destroying a session

So let's make a Sessions Controller (note that there's no corresponding Sessions model!)

Making a sessions controller

We can create a sessions controller via '`ruby script/generate controller sessions`'.

sessions_controller.rb

```
def new
  unless User.count > 0
    flash[:notice] = "Please create the first user"
    redirect_to new_user_path
  end
end

def create
  self.current_user = User.authenticate(params[:email],
  params[:password])
  unless logged_in?
    flash[:notice] = "Incorrect login/password"
    render :action => 'new' and return
  end
  redirect_to(root_path)
end

def destroy
  reset_session
  flash[:notice] = "You've been logged out"
  redirect_to(root_path)
end
```

We're using some methods we didn't define in this controller (User.authenticate, logged_in?). We'll come back to these.

In particular, we aren't actually trying to find the user and set the session variable in the controller; e.g. we didn't try to write:

```
def create
  user = User.find_by_email(params[:email])
  unless user && user.crypted_password ==
  user.encrypt(params[:password])
    flash[:notice] = "Incorrect login/password"
    render :action => 'new' and return
  end
  session[:user_id] = user.id
  redirect_to(root_path)
end
```

Why not?

Skinny controllers, Fat models

More Rails philosophy: This kind of code doesn't belong in our controller.

Let the model figure out whether a user meets the conditions of being logged in. The controller is only supposed to figure out what view to show you in that case.

Implementing the methods

In the User model, add these methods:

```
def self.authenticate(email, password)
  user = find_by_email(email)
  user && user.authenticated_by?(password) ?
  user : nil
end

def authenticated_by?(password)
  encrypt(password) == crypted_password
end
```

To app/controllers/application.rb we add

```
def logged_in?
  current_user != :false
end
```

But wait…

Where is the current user coming from? We used this in the sessions controller too.

```
self.current_user =
  User.authenticate(params[:email],
  params[:password])
```

There's no current_user instance variable.

Getting/setting the current user

To do this, we fake it by adding methods to the app/controllers/application.rb.

```
def current_user=(user)
  session[:user_id] = user.nil? ? nil : user.id
  @current_user = user || :false
end

def current_user
  @current_user ||= (login_from_session || :false)
end

protected
  def login_from_session
    self.current_user = User.find(session[:user_id]) if
   session[:user_id]
end
```

Adding helpers to views

If we want to be able to use the `logged_in?` and `current_user` methods in our views, we can do this by declaring

```
helper_method :current_user,
:logged_in?
```

in app/controller/application.rb.

Allowing for logins

Routing logins

Let's set up routes in config/routes.rb to allow for logins:

```
map.resources :sessions
map.login 'login', :controller =>
   'sessions', :action => 'new'
map.logout 'logout', :controller =>
   'sessions', :action => 'destroy'
```

We create *named* routes; these will match the URLs /login and /logout, but will also let us refer to login_path in our views.

An application view

Now we create an application layout in app/views/layouts/application.html.erb (and deleting the other layouts).

application.html.erb

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="content-type" content="text/html;charset=UTF-8" />
  <title>Sample Blog<%= yield :title %></title>
  <%= stylesheet_link_tag 'scaffold' %>
</head>
<body>

<p><% if logged_in? -%>
  You are: <%= current_user.name %>
  <%= link_to('logout', logout_path) %> |
  <%= link_to_unless_current('user mgmt', users_path) %> |
<% else %>
  <%= link_to('login', login_path) %> |
<% end -%>
<%= link_to_unless_current('all posts', posts_path) %>
</p>
<hr />

<p style="color: green"><%= flash[:notice] %></p>

<%= yield  %>

</body>
</html>
```

Now give this all a try…

Want to get a job or internship in Information Science?

- Is your resume ready for companies to see, or would you like it to help you get that job?
- The ISSA will be having a Resume Workshop specifically tailored to Information Science with Craig Jones from Cornell Career Services
- Monday, October 6th, 2008 @ 4:30 in Upson 205
- We'll have food and prizes!

infoscience STUDENT ASSOCIATION

Announcement

Last class for INFO 2310 will be November 7, not October 31.