Worksheet 3: Modeling users – part1

CSC 340

In this worksheet, we will create a users model that will include a username, email and password. This model will be used to authenticate users to the site (web application). Create a new rails application and do the following:

1. ~~Let’s create a User model and controller.~~ 
   1. ~~Generate a controller called Users with an action called new.~~
   2. ~~Generate a model called User with fields name(of type string) and email (of type string)~~
   3. ~~Apply the migration (to undo a migration, you can use the command: rails db:rollback )~~
   4. ~~Switch to the console or sqlite view and check your model~~
2. ~~Adding validations~~
   1. ~~In the user.rb file, the model file, add validations for the presence of the name, email, length, format…. Add the lines:~~

~~before\_save {email.downcase! }~~

~~#this forces the email to be converted to lowercase before it is saved to the database to enforce uniqueness.~~

~~validates :name, presence: true, length: {maximum: 50}~~

~~#makes sure the name is present and sets a maximum length~~

~~VALID\_EMAIL\_REGEX = /\A[\w+\-.]+@[a-z\d\-.]+\.[a-z]+\z/i~~

~~validates :email, presence: true, length: {maximum: 250}, format: {with: VALID\_EMAIL\_REGEX }, uniqueness: {case\_sensitive: false}~~

~~#makes sure that the email is present, sets a maximum length and makes sure the format is appropriate and ensures the uniqueness is not case sensitive (for example: Jk and jk will be considered the same)~~

* 1. ~~Adding the database index: if a column in your database can be used to find records (in our case, we can find users by their email address), we can create a database index to make searching for a particular record easier (think of a database index as being similar to a book index – it creates a reference to the record). To generate an index, we need to generate a new rails migration:~~

~~rails generate migration add\_index\_to\_users\_email~~

* 1. ~~Apply the migration.~~
  2. ~~Test your model by switching to the rails console or sqlite3 view~~

1. ~~Adding a secure password: We will use the rails method has\_secure\_password, which will add the ability to have the password field (which we’ll call password\_digest) hashed, creates virtual attributes and an authenticate method that returns the user when the login/password is correct.~~ 
   1. ~~First, we add a new field in the User model called password\_digest, we can do this using a migration:~~

~~rails generate migration add\_password\_digest\_to\_users password\_digest:string~~

* 1. ~~Apply the migration~~
  2. ~~Open the application’s Gemfile and right after the line that reads:~~

~~gem ‘rails’, ‘5.0.0.1’ , add the line:~~

~~gem ‘bcrypt’, ‘3.1.11’~~

~~#this allows has\_secure\_password to use the hash function bcrypt, which is used to hash the password so that it is stored hashed and not in plaintext.~~

* 1. ~~Type: bundle install~~

~~#This will installs all the gems that have been added to the Gemfile file.~~

1. ~~Adding the password~~
   1. ~~In the user.rb file, the User model, after all the validations, add the line:~~

~~has\_secure\_password~~

~~#this adds the method to the User model so we can use it. Then add the line:~~

~~validates :password, presence: true, length: {minimum: 6}~~

1. ~~Testing your User mode.~~

~~Switch to the console view and try to create a new User object, try commands like:~~

~~User.create(name: “Khadija”, email:~~ [~~“khadija@gmail.com~~](mailto:)~~”, password: “password”, password\_confirmation: “password”)~~

~~u = User.find\_by(email: “~~[~~Khadija@gmail.com~~](mailto:Khadija@gmail.com)~~”)~~

~~u.password\_digest~~

~~u.autheticate(“kjsldjf”)~~

~~u.authenticate(“password”)~~

~~#notice how the password digest field is stored.~~

1. ~~Creating the show action/view~~
   1. ~~Create a show action in the Users controller and its corresponding view. In the show view, you can display the current user’s name and email address:~~

~~<% provide(:title, @user.name) %>~~

~~<%= @user.name %>, <%= @user.email %>~~

* 1. ~~In the Users controller, in the show action add the following line:~~

~~@user = User.find(params[:id])~~

~~# when this action is called, a parameter is passed to it, the parameter is the id of a user. The find method finds the particular user with that id and returns it and stores it in @user.~~

1. ~~Adding a Gravatar image to the model.~~ 
   1. ~~To create a gravatar, go to:~~ [~~http://en.gravatar.com/~~](http://en.gravatar.com/)~~, click on create your own gravatar, enter the email address you want the gravatar associated with along with a username and password, you’ll receive an email and be asked to proceed from there to add a new picture.~~
   2. ~~Update the show view by displaying the gravatar of the user and its name rather than its email address:~~

~~<h1>~~

~~<%= gravatar\_for @user %>~~

~~<%= @user.name %>~~

~~</h1>~~

* 1. ~~We need to define the gravatar\_for in the app/helpers/users\_helper.rb file. gravatar\_for is a helper and needs to be defined in the helpers folder which will make it available to all views. Methods defined in any helper file are available to all views.~~

~~module UsersHelper~~

~~def gravatar\_for(user, size: 80)~~

~~gravatar\_id = Digest::MD5::hexdigest(user.email.downcase)~~

~~gravatar\_url = “[https://secure.gravatar.com/avatar/#{gravatar\_id}](https://secure.gravatar.com/avatar/" \l "{gravatar_id})?s=#{size}”~~

~~image\_tag(gravatar\_url, alt: user.name, class: "gravatar")~~

~~end~~

~~end~~

~~#In this helper, a gravatar\_id is generated by taking the hash (MD5 hash) of the email address of the user (converted to lowercase to avoid having duplicate email addresses with different cases). Then a url is generated to connect to the gravatar. Finally, an image is generated (using the image\_tag helper), the image’s source is the gravatar’s url generated using the gravatar id. The image has a class that is called “gravatar” to style it.~~

* 1. ~~Create a route for your Users controller by adding the following line in config/routes.rb file:~~

~~resources :users~~

* 1. ~~Start the server and try your application by adding the following to the browser’s url: /users/1~~

~~#users links to the Users controller, then it is followed by an id, which is then passed as a parameter to the show view to display the User object with that corresponding id. In our example, I picked id 1 (change the id to display different User objects).~~

~~Worksheet 4: Modeling users – part2~~

~~CSC 340~~

~~In this worksheet, we will create the signup form for our site. This will create a web interface for users to sign into the app.~~

1. ~~We will use the rails form\_for helper, which takes as a parameter the active record object and builds a form from the object’s attributes.~~ 
   1. ~~We will route the signup page through the new action in the Users controller.~~
   2. ~~First, create a new User object in the new action called @user.~~
   3. ~~Next, open the new view (new.html.erb) and let’s create the form elements.~~

~~<% provide(:title, ‘Sign up’) %>~~

~~<h1> Sign up </h1>~~

~~<%= form\_for(@user) do |f| %>~~

~~<%= f.label :name %> <br />~~

~~<%= f.text\_field :name %><br />~~

~~<%= f.label :email %><br />~~

~~<%= f.text\_field :name %><br />~~

~~<%= f.label :password %><br />~~

~~<%= f.password\_field :password %><br />~~

~~<%= f.label :password\_confirmation, “Confirmation” %><br />~~

~~<%= f.password\_field :password\_confirmation %><br />~~

~~<%= f.submit “Create my account” %>~~

~~<% end %>~~

* 1. ~~Now, let’s add some CSS to style the form. Create a new stylesheet in the app/assets/stylesheets/ directory, call it custom.scss and add the following:~~

~~/\* forms \*/~~

~~input, textarea, select, .uneditable-input {~~

~~border: 1px solid blue;~~

~~width: 70%;~~

~~margin-bottom: 15px;~~

~~@include box\_sizing;~~

~~}~~

~~input {~~

~~height: auto;~~

~~}~~

* 1. ~~Next, let’s create the action that is called when the user presses the submit button. We will write a create action that will use the post request to submit the user’s input. The create action should be added to the users controller (app/controllers/users\_controller.rb)~~

~~def create~~

~~@user = User.new(user\_params)~~

~~#create a new user object with the parameters sent by the form~~

~~if @user.save~~

~~flash[:success] = “Account created successfully!”~~

~~#if the User object saves successfully~~

~~redirect\_to @user #redirects to the show action – executes the action and renders view~~

~~else~~

~~render ‘new’~~

~~#this will just render the new view if the user is not saved successfully (this usually means that the validations didn’t pass)~~

~~end~~

~~end~~

~~private~~

~~def user\_params~~

~~params.require(:user).permit(:name, :email, :password,~~

~~:password\_confirmation)~~

~~#:user is the required attribute while name, email, password and password\_confirmation are permitted to pass to the new action~~

~~end~~

* 1. ~~Now, we need to allow the flash messages to display on the template view. In app/views/layouts/application.html.erb, add the following:~~

~~Inside the body element, add:~~

~~<div class=”container”>~~

~~<% flash.each do |message\_type, message| %>~~

~~<div class= “alert alert-<%= message\_type %>”><%=message %></div>~~

~~<% end %>~~

~~<%= yield %>~~

~~</div>~~

* 1. ~~Add the route: get ‘/signup’, to: ‘users#new’~~

~~\* Railstutorial.org~~

~~Worksheet 5: Modeling users – part3~~

~~CSC 340~~

~~In this worksheet, we will create the basic login and logout for our application.~~

1. ~~First, we’ll create a session for our application. Since HTTP is stateless, each request is treated as independent from any other request, so we will be using a session to remember a user’s identity from one request/page to another. We will implement a session using cookies (small pieces of text/information that are stored on the user’s browser)~~
2. ~~The cookies will be able to store the user id so that a user can go from page to page and still be remembered by the application. We will therefore use the rails method session to create temporary sessions that last until the browser closes (in which case they expire).~~
3. ~~We will creae a Sessions controller, a login form and several actions.~~ 
   1. ~~Generate a controller called Sessions with one action called new~~
   2. ~~Next, let’s generate the routes for new and the other actions we will create. We need to generate a get route for sessions#new called login (get ‘/login’, to: ‘sessions#new’ ) and a post route for sessions#create called login and a delete route to sessions#destroy called logout.~~
   3. ~~Next, let’s generate our login view (in the app/views/sessions/new.html.erb file):~~

~~<% 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>~~

~~#Here, note the use of several bootstrap style classes.~~

1. ~~Authenticating a user:~~
   1. ~~To log a user in, we’ll use a cookie that will automatically expire when the browser is closed. We therefore need to use several methods across several controllers and actions, we will define those methods in a helper.~~ 
      1. ~~We will include the helper module in the base of all controllers, in the app/controllers/application\_controller.rb file add the following before the “end”:~~

~~include SessionsHelper~~

* + 1. ~~In the app/helpers/sessions\_helper.rb file, add the following methods that will be used over and over again:~~

~~def log\_in(user)~~

~~session[:user\_id] = user.id #saves the id of the user to the session hash~~

~~end~~

~~def current\_user~~

~~@current\_user ||= User.find\_by(id: session[:user\_id])~~

~~#this could also be written as:~~

~~#@current\_user = @current\_user || User.find\_by(id: session[:user\_id])~~

~~#this means that if the @current\_user is not nil, return it, otherwise, assign to it the User object with the specified id.~~

~~end~~

~~def logged\_in?~~

~~!current\_user.nil? #returns true if the user is logged in and false otherwise.~~

~~end~~

~~#logout current user by deleting its id from the session and setting @current\_user to nil~~

~~def log\_out~~

~~session.delete(:user\_id)~~

~~@current\_user = nil~~

~~end~~

* 1. ~~Let’s add create and destroy actions in the Sessions controller.~~

~~def create~~

~~user = User.find\_by(email: params[:session][:email].downcase) #finds a user with the supplied email~~

~~if user && user.authenticate(params[:session][:password]) #if the user is authenticated – we used this in part 2 as well~~

~~log\_in user #use the helper to log the user in by saving its id to the session hash~~

~~redirect\_to user #redirects the user’s view~~

~~else~~

~~flash.now[:danger] = ‘Invalid email/password combination’~~

~~render ‘new’ #if authenticate fails, render the new view again so the user can try to login again~~

~~end~~

~~end~~

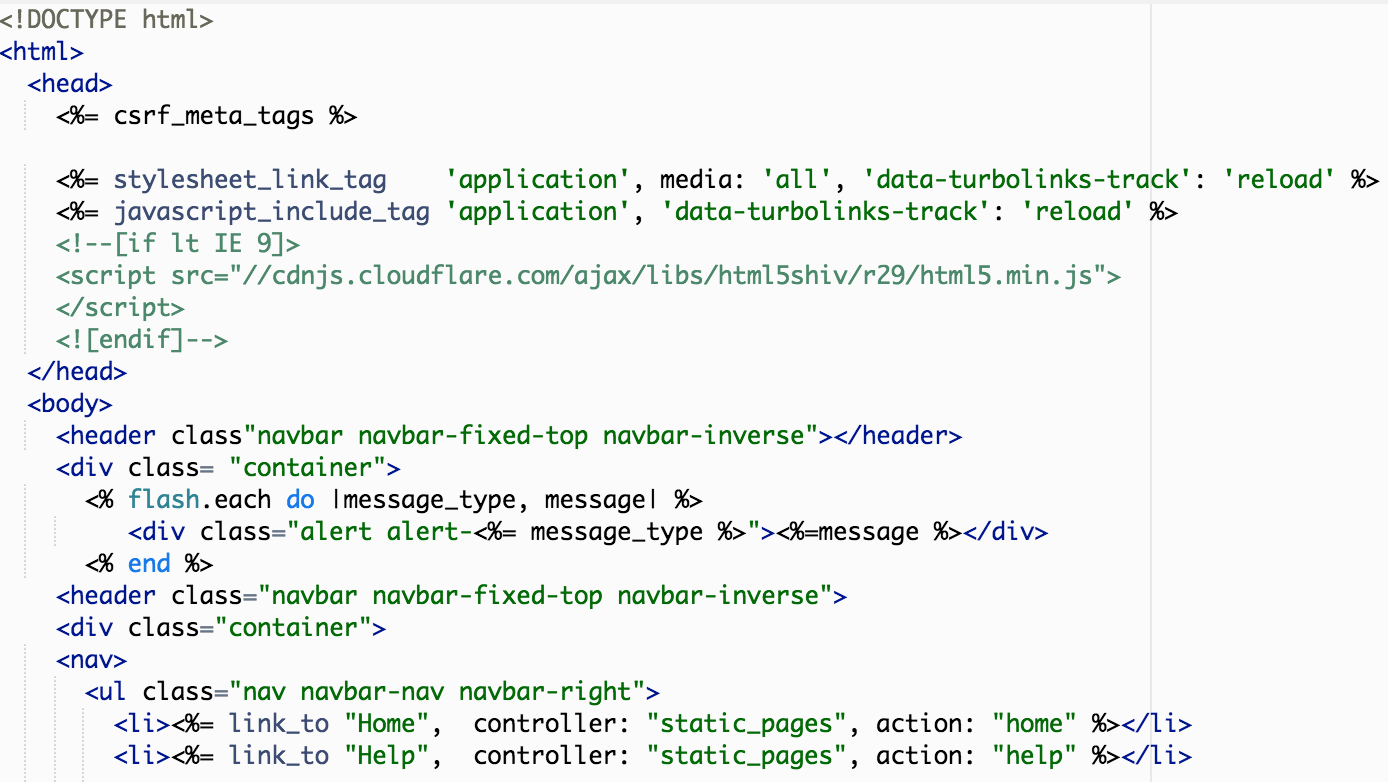
~~def destroy~~

~~log\_out~~

~~redirect\_to root\_url~~

~~end~~

1. Now, let’s add some functional links to the website. In the application.html.erb, fill in the links in the nav bar as follows:





* 1. ~~Now, we need to add bootstrap in order to make user of the drop down menu feature. Add the following to the /app/assets/javascripts/application.js file, right after //= require jquery\_ujs:~~

~~//= require bootstrap~~

1. You can create a static controller called static\_pages with actions: home, help and about

~~Worksheet 6: Modeling users – part4~~

~~CSC 340~~

~~In this worksheet, we will work on updating and deleting users. We will include an admin field in the User model to indicate when a user is an admin. We will also create filters to limit/control access to some actions/views.~~

1. ~~Enabling users to login right after they signup:~~
   1. ~~In the users\_controller.rb file, in the create action, add the following line after: if @user.save~~

~~log\_in @user #this will log the user in right after they signup for an account.~~

1. ~~Next, we’ll work on updating, showing and deleting User objects. We will therefore add the edit, update, index, destroy actions.~~ 
   1. ~~Updating users~~
      1. ~~Let’s create an edit action in the Users controller. In the edit action, the @user instance field is assigned a User object using the find(params[:id])~~
      2. ~~Next, we need to create an edit view for the action, the edit view will use some of the same form elements as the new view. To consolidate both views, we will create a partial that will be used by both views (so that code will not be repeated). Create a partial called \_form.html.erb in the /app/views/users/ folder.~~

~~<%= form\_for(@user) do |f| %>~~

~~<%= render ‘shared/error\_messages’, object: @user %>~~

~~<%= 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-contorl’ %>~~

~~<%= f.label :password\_confirmation, “Confirmation” %>~~

~~<%= f.password\_field :password\_confirmation, class: ‘form-contorl’ %>~~

~~<%= f.submit yield(:button\_text) , class: “btn btn-primary” %>~~

~~<% end %>~~

* + 1. ~~Modify the new.html.erb view to use the partial, it should be as follows:~~

~~<% provide(:title, “Sign up”) %>~~

~~<% provide(:botton\_text, ‘Create my account’) %>~~

~~<h1>Sign up</h1>~~

~~<div class=”row”>~~

~~<div class=”col-md-6 col-md-offset-3”>~~

~~<%= render ‘form’ %>~~

~~</div>~~

~~</div>~~

* + 1. ~~Now, let’s create the edit.html.erb view as follows:~~

~~<% provide(:title, ‘Edit user’) %>~~

~~<% provide(:button\_text, ‘Save changes’) %>~~

~~<h1>Update your profile</h1>~~

~~<div class=”row”>~~

~~<div class=”col-md-6 col-md-offset-3”>~~

~~<%= render ‘form’ %>~~

~~<div class=”gravatar\_edit”>~~

~~<%= gravatar\_for @user %>~~

~~<a href= “~~[~~http://gravatar.com/emails~~](http://gravatar.com/emails) ~~“ target=”\_blank”>change</a>~~

~~</div>~~

~~</div>~~

~~</div>~~

* + 1. ~~Now, let’s update the link to the edit action. In the layout page (app/views/application.html.erb), you can modify the line that links to “Settings” by adding the path to the edit action/view, replace the ‘#’ by: edit\_user\_path(current\_user) .~~
    2. ~~Let’s create an update action with the following:~~

~~def update~~

~~@user = User.find(params[:id]) #this finds the user with the specified id~~

~~if @user.update\_attributes(user\_params) #if the user’s attributes are updated successfully~~

~~flash[:success] = “Profile updated”~~

~~redirect\_to @user #redirect the user to the show action/view.~~

~~else~~

~~render ‘edit’ #if the edit was unsuccessful, render the edit view again~~

~~end~~

~~end~~

* 1. ~~Creating filters to limit access:~~
     1. ~~In the users\_controller.rb file, add a before filter that requires only logged in users to access the edit and update view. At the top of the file, add:~~

~~before\_action :logged\_in\_user, only: [:edit, :update]~~

* + 1. ~~We also need to add another filter that calls a new action correct\_user. Create a filter identical to the one above except that it needs to call the action correct\_user to make sure that the correct user is logged in.~~
    2. ~~Then, at the end of the file, add a definition for the logged\_in\_user action as follows. This action is called before the edit and update actions are called, if the user is not currently logged in, a message appears and the user is redirected to the login page.~~

~~def logged\_in\_user~~

~~write code that calls the logged\_in? action (defined in a helper). If logged\_in? returns false, create an appropriate flash message then redirect the user using the redirect\_to helper to login\_url.~~

~~end~~

* + 1. ~~Now, it is time to define the correct\_user action. Define the correct\_user action that confirms that this is the correct user as follows:~~

~~def correct\_user~~

~~@user = User.find(params[:id])~~

~~redirect\_to(root\_url) unless current\_user?(@user)~~

~~#end~~

~~end~~

* + 1. ~~Now we need to define current\_user?(user) action in the sessions helper.~~

~~def current\_user?(user)~~

~~user == current\_user #returns true if the parameter is the current user that is logged in.~~

~~end~~

* 1. ~~Now, we will work on showing all the users that have created accounts in the application~~
     1. ~~First, let’s create an index action in the users controller. The index view will allow us to display information about all the users logged in the application.~~

~~def index~~

~~@users = User.all #this will return all the users and store them in @users~~

~~end~~

* + 1. ~~Update the first before\_action filter (the one that called the logged\_in\_user action) by adding the index action to the list that contains the edit and update actions.~~
    2. Update the layout view (the app/views/layouts/application.html.erb file) by removing the ‘#’ for the link\_to “Users” and replacing it with users\_path, which will connect to the index action/view.
    3. ~~Let’s work on the index.html.erb view, see below:~~

~~<% provide(:title, ‘All users’) %>~~

~~<h1>all users </h1>~~

~~<ul class=”users”>~~

~~<%= @users.each do |user| %>~~

~~<li>~~

~~<%= gravatar\_for user, size: 50 %>~~

~~<%= link\_to user.name, user %>~~

~~</li>~~

~~<% end %>~~

~~</ul>~~

* 1. ~~Deleting users and Admin users. Now we need to add functionality to be able to delete a user. This functionality however, should only be available to administrators of the site.~~ 
     1. ~~First, we will add a field in the User model that can specify whether the user is an administrator or not. To achieve this, we need to generate a new migration to the database that adds the admin column as a Boolean to the users table:~~

~~rails generate migration add\_admin\_to\_users admin:boolean~~

* + 1. ~~Open the migration file in the db/migrate directory, make sure to specify that the default value for admin should be false.~~

~~def change~~

~~add\_column :users, :admin, :boolean~~ **~~, default: false~~**

~~end~~

* + 1. Apply the migration.
    2. Next, let’s create a destroy action which will be called to delete User objects. In the users\_controller.rb file, add the following:

def destroy

User.find(params[:id]).destroy

flash[:success] = “User successfully deleted”

redirect\_to users\_url

ends

* + 1. Make sure to also add the delete method to the list of methods in the before filter, :logged\_in\_user.
    2. You also need another filter that ensures only an admin user is able to use the delete action. The filter should be as follows:

before\_action :admin\_user, only: :destroy

* + 1. Add the admin\_user action to the Users controller under private (i.e., this is a private method).

def admin\_user

redirect\_to(root\_url) unless current\_user.admin? #this makes sure that if the user is not an admin user, they are redirected to root\_url

end

* + 1. In the index view, add the following lines to allow only admins to view the link to delete a user (note here that an admin can not delete their own account). Add these lines after you display the user.name.

if current\_user.admin? && !current\_user?(user) %>

| <%= link\_to “delete”, user, method: :delete, data: {confirm: “you sure?”} %>

* + 1. On the console view, create a User object with the admin field true. Then test your application. (If your console is not working, create the admin user in the new action of the Users controller).