CS 416
Web Programming

Ruby on RAILS
Chapter 6-7 User sign up and modeling

Dr. Williams
Central Connecticut State University

A look ahead

- Look at building parts of scaffolding by hand -- how do the parts fit together
  - Create controller and view
  - Create model
- Working with model
- Validations
- Password hashing

Note full CSS for lectures now on class GitHub:

https://github.com/CCSU-CS416F17/CS416F17CourseInfo/blob/master/custom.scss

Create controller and sign up page

- Generate User controller and *new* view

```
rails generate controller Users new
```

Note: plural name for controller

- Change route to new view to */signup*

```
get  '/signup',  to: 'users#new'
```

- Update link path on home page: ***signup_path***

- Update stub view to provide title to layout

```
<% provide(:title, 'Sign up') %>
```

Create User model

- Generate User model

```
rails generate model User name:string
email:string
```

Note: singular name for model

- And migrate

```
rails db:migrate
```

- Note: Can view data in SQLite using DB Browser for SQLite (http://sqlitebrowser.org/)

Working with Model

```
rails console --sandbox
```

- Create instance without saving

```
user = User.new(name: "Chad", email: "chad@ccsu.edu")
user.save
```

- Create instance and save

```
user = User.create(name: "Chad", email: "chad@test.org")
```

- Before save, rails check if valid: `user.valid?`

- Delete instance from DB

```
user.destroy
```

Finding Model elements

- Find by primary key

```
User.find(1)
```

- Find by specific attribute

```
User.find_by(email: "chad@ccsu.edu")
```

- Find first record

```
User.first
```

- Wild card

```
User.where(["name LIKE ?","C%"])
```

- Retrieve all

```
User.all
```

Updating model

- Retrieve model, make change, save

```
user = User.find(1)
user.name = "Bob"
user.save
```

- Update and save all at once – and checks valid

```
user = User.find(1)
user.update_attributes(name: "Will",
      email: "will@test.org")
```

Validations - presence

- Presence – value and not blank

```
class User < ApplicationRecord
  validates :name, presence: true
end
```

- Test and see validity errors

```
user = User.new(name: "  ", email: "c@c.com")
user.valid?   (returns false)
user.errors.full_messages
user.save     (returns false)
```

- **Add check for email presence to model**

Validations - length

- Length conditions

```
validates :name, length: { maximum: 50 }
validates :zipcode, length: { minimum: 5 }
validates :login, length: { in: 5..12 }
validates :state, length: { is: 2 }
```

Can add multiple conditions to each field

```
validates :name,  presence: true, length: { maximum: 50 }
validates :email, presence: true, length: { maximum: 255 }
```

Validations - format

- Format validations can be specified by regular expressions

Note: naming standard for constants in Ruby all upper case

```
VALID_EMAIL_REGEX = /\A[\w+\-.]+@[a-
z\d\-.]+\.[a-z]+\z/i

validates :email, presence: true,
length: { maximum: 255 },
format: { with: VALID_EMAIL_REGEX }
```

Try http://www.rubular.com/ for creating/debugging regular expressions

Validations - uniqueness

- Checks to guarantee uniqueness

```
validates :email, uniqueness: true
```

Or

```
validates :email,
    uniqueness: { case_sensitive: false}
```

(Enforces in Rails app, but does not enforce in DB)

Step 1 add:

```
before_save {email.downcase! }
```

Database indices

- In large DB doing a find on any string can be very costly unless it is indexed – *full table scan*
- Indexing allows a lookup to be done very quickly and also can enforce uniqueness at DB level
- To create index must create migration by hand

```
rails generate migration add_index_to_users_email
```

- Now add index to the migration

```
class AddIndexToUsersEmail < ActiveRecord::Migration[5.0]
  def change
    add_index :users, :email, unique: true
  end
end
```

- Then migrate

```
rails db:migrate
```

- **Delete test *fixture* users** in test/fixtures/users.yml, then migrate test

```
rails db:migrate RAILS_ENV=test
```

Note if you have data which doesn't comply with the index in either environment add `db:reset` *before* `db:migrate` *to delete the data first*

Password security

Storing passwords

- **You should NEVER EVER store a password in plaintext in the database**

- The right way to store any password is to instead store a *secure hash* of their password

- A *secure hash* is a one way function that ensure that from the hash no one can ever recover the original password, but your can quickly verify if the password is provided its hash matches the one stored

Adding a secure password

- Rails has built in functionality to help. Adding to your model:

```
class User < ApplicationRecord
  ...
  has_secure_password
end
```

This adds:

- Ability to save securely hashed `password_digest`
- Pair of virtual attributes: `password`, and `password_confirmation` that must be present and match for the model to be `valid`
- An `authenticate` method that returns the user when the password is correct, otherwise false.

Add password_digest

- For the has_secure_password to work we need a password_digest column on the table, so…
- Create a generation for it

```
rails generate migration
add_password_digest_to_users password_digest:string
```

  - The standard naming allows rails to figure out which model and what field to add

```
class AddPasswordDigestToUsers <
ActiveRecord::Migration[5.0]
  def change
    add_column :users, :password_digest, :string
  end
end
```

- Then migrate

```
rails db:migrate
```

Add security gems

- Add bcrypt to Gemfile

```
gem 'bcrypt',                '3.1.11'

~~~~~~~~~~~~~~
bundle install
```

Using the has_secure_password funcionality

- Now rather than:

user = User.new(name: "Chad" , email: "c@c.com")
Yields `user.valid? => false`

- Now password must be provided and match:

user = User.new(name: "Chad" , email: "c@c.com",
password: "pass",
password_confirmation: "pass")

Yields `user.valid? => true`

Add presence and length restrictions to model

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

User password usage - **authenticate**

- Now when user is retrieved from DB, can authenticate against password

```
User.create(name: "Chad", email: "C@chad.com", password:
"password", password_confirmation: "password")

user = User.first

User.authenticate("wrong")
=> False

user.authenticate("password")
=> #<User id: 2, name: "Chad", email: "c@chad.com",
created_at: "2016-11-16 05:46:24", updated_at: "2016-11-16
05:46:24", password_digest:
"$2a$10$scxQ6XtLYQvLtAxEPFpyze9tpuqyj5Uvj1LPK2MLtqj...">
```

Adding debug information

- Take advantage of common page template and knowledge of environment to enable debug info

- In *application.html.erb*

```
<%= debug(params) if Rails.env.development? %>
```

Create REST operations

```
Rails.application.routes.draw do
  root 'static_pages#home'
  get  '/help',    to: 'static_pages#help'
  get  '/about',   to: 'static_pages#about'
  get  '/contact', to: 'static_pages#contact'
  get  '/signup',  to: 'users#new'
  resources :users
end
```

Create simple *show* view/controller

rails routes:

HTTP request	URL	Action	Named route
GET	/users	index	users_path
GET	/users/**:id(.:format)**	show	user_path(user)

- View app/views/users/show.html.erb

```
<% provide(:title, @user.name) %>
<h1>
  <%= @user.name %>, <%= @user.email %>
</h1>
```

- Add show method to controller

```
def show
   @user = User.find(params[:id])
end
```

Creating sign up form

- Create a new user object to pass to form to be populated

```
class UsersController < ApplicationController

  def show
    @user = User.find(params[:id])
  end

  def new
    @user = User.new
  end
end
```

The sign up form

F16 early

```
<% provide(:title, 'Sign up') %>
<h1>Sign up</h1>

<div class="row">
  <div class="col-md-6 col-md-offset-3">
    <%= form_for(@user) do |f| %>
      <%= f.label :name %>
      <%= f.text_field :name %>

      <%= f.label :email %>
      <%= f.email_field :email %>

      <%= f.label :password %>
      <%= f.password_field :password %>

      <%= f.label :password_confirmation, "Confirmation" %>
      <%= f.password_field :password_confirmation %>

      <%= f.submit "Create my account", class: "btn btn-primary" %>
    <% end %>
  </div>
</div>
```

form_for

```
<%= form_for(@user) do |f| %>
      <%= f.label :name %>
      <%= f.text_field :name %>
    …
<% end %>
```

Tells rails to generate code that is specifically designed to assign the passed fields to the object specified in the form_for

REST create action

- Route

POST /users(.:format) users#create

- Populating from form

```
def create
  @user = User.new(...)
  if @user.save
    # Handle a successful save.
  else
    render 'new'
  end
end
```

Strong parameters

- In previous rails this would work:

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

Problem is just like concept of SQL injection, malicious user can write to any user field they want

- Correct way for using user input is ***strong parameters***
  - Explicitly state parameters expected and which allowed

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

Strong parameters cont.

- Revised create with strong parameters

```
def create
  @user = User.new(user_params)
  if @user.save
    # Handle a successful save.
  else
    render 'new'
  end
end

private

  def user_params
    params.require(:user).permit(:name, :email,
        :password, :password_confirmation)
  end
```

Displaying errors

- Create errors partial:

app/views/shared/_error_messages.html.erb

```
<% if @user.errors.any? %>
  <div id="error_explanation">
    <div class="alert alert-danger">
      The form contains <%= pluralize(@user.errors.count, "error") %>.
    </div>
    <ul>
    <% @user.errors.full_messages.each do |msg| %>
      <li><%= msg %></li>
    <% end %>
    </ul>
  </div>
<% end %>
```

Include in new page

<%= render 'shared/error_messages' %>

Revised new view – with error messages and style

```
<% provide(:title, 'Sign up') %>
<h1>Sign up</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 "Create my account", class: "btn btn-primary" %>
    <% end %>
  </div>
</div>
```

Adding sign up create for POST

Add to routes:

```
post '/signup',  to: 'users#create'
```

Modify post location of new form:

```
<%= form_for(@user, url: signup_path) do |f| %>
```

Update success path:

```
def create
    @user = User.new(user_params)
    if @user.save
      flash[:success] = "Welcome to the Sample App!"
      redirect_to @user
    else
      render 'new'
    end
  end
```

Flash messages

```
flash[:success] = "Welcome to the Sample App!"
```

- *Flash* is a built in mechanism in Rails to show a message on a subsequent page, but have it disappear on refresh or visiting a 2nd page
- Common flash types:
  - :success
  - :info
  - :warning
  - :danger

Add display of flash messages to site layout

```
<div class="container">
  <% flash.each do |message_type, message| %>
    <div class="alert alert-<%= message_type %>">
      <%= message %>
    </div>
  <% end %>
  <%= yield %>
  <%= render 'layouts/footer' %>
  <%= debug(params) if Rails.env.development? %>
</div>
```

Force SSL in production

- *config/environments/production.rb*

```
# Force all access to the app over SSL,
# use Strict-Transport-Security,
# and use secure cookies.
config.force_ssl = true
```

Push to Heroku

Rails debugger

- Add debugger line to drop server into debugging – like a breakpoint

```
def show
   @user = User.find(params[:id])
   debugger
end
```

- When breakpoint is reached brings up byebug prompt

```
(byebug)
```

- Can interact just like rails console inspect and even change data, when done press Ctrl-D

Byebug quick reference

- Highly recommend:

http://fleeblewidget.co.uk/2014/05/byebug-cheatsheet/

To make break point conditional:

```
byebug if foo == "bar"
```

n[ext] <number>

Go to next line, stepping over function calls. If number specified, go forward that number of lines.

s[tep] <number>

Go to next line, stepping into function calls. If numberis specified, make that many steps.

up <number> / down <number>

Step up/down stack trace.

w[here]

Display full stack trace.

h[elp] <command-name>

Get help. With no arguments, returns a list of all the commands Byebug accepts. When passed the name of a command, gives help on using that command.

Creating helper methods

- Add Gravatar (http://en.gravatar.com/) image to show page

```
<%= gravatar_for @user %>
```

- Each controller automatically generates corresponding helper class: *app/helpers/users_helper.rb*
- Add gravatar_for method:

```
# Returns the Gravatar for the given user.
def gravatar_for(user)
  gravatar_id = Digest::MD5::hexdigest(user.email.downcase)
  gravatar_url =
     "https://secure.gravatar.com/avatar/#{gravatar_id}"
  image_tag(gravatar_url, alt: user.name, class: "gravatar")
end
```

Try example@railstutorial.org