

CS 142 Section 5

Forms and Sessions

Overview

1. Forms
2. Validation
3. Uploading
4. Virtual Attributes
5. Session
6. Flash
7. Hashing
8. Filters

Review: RESTful Path Mapping

Controller#Action routed to	Path	HTTP Verb	What it should do
PhotosController#index	/photos	GET	Show all Photo objects
PhotosController#show	/photos/<id>	GET	Show Photo object with specified id
PhotosController#edit	/photo/<id>/edit	GET	Edit Photo object with specified id
PhotosController#create	/photos	POST	Create new Photos object
PhotosController#update	/photos/<id>	PATCH	Update an existing object

More path mappings: <http://guides.rubyonrails.org/routing.html#crud-verbs-and-actions>

Form For

Existing instance variable of model
from database that the form is for.

```
<% form_for(@student) do |form| %>
```

```
  <table class="form">
```

```
    <tr>
```

```
      <td class="label">Name:</td>
```

```
      <td><%= form.text_field(:name) %></td>
```

```
    </tr>
```

```
    <tr>
```

```
      <td class="label">Date of birth:</td>
```

```
      <td><%= form.text_field(:birth) %></td>
```

```
    </tr>
```

```
    ...
```

```
  </table>
```

```
  <%= form.submit %>
```

```
<% end %>
```

Initial value will be
@student.name

Form For HTML

Correct action and method

(if `@student.new_record?` is false).

Assuming user with id 1.



```
<form accept-charset="UTF-8" action="/students/1" class="
edit_student" id="edit_student_1" method="post">
<input name="_method" type="hidden" value="patch" />
  <table class="form">
    <tr>
      <td class="label">Name:</td>
      <td><input id="student_name" name="student[name]"
        type="text" value="Anderson"/></td>
    </tr>
    <tr>
      <td class="label">Date of birth:</td>
      <td><input id="student_birth" name="student[birth]"
        type="text" value="1987-10-22"/></td>
    </tr>
    ...
  </table>
  <input name="commit" type="submit"
    value="Update Student" />
</form>
```

Initial value for the
updating student!

Default text from
form.submit

Review: Post Action Method (Update)

```
def update
  @student = Student.find(params[:id])
  if @student.update_attributes(student_params(params[:
student]))
    redirect_to @student
  else
    render(action: :edit)
  end
end
```

Redirects on success

Hash with all of form data and
can access individual values.
Ex: `params[:student][:birth]`

```
private
def student_params(params)
  return params.permit(:name, :birth, :gpa, :grad)
end
```

Rails 4.0+ update pattern “strong parameters”. Must tell Rails which attributes are **permitted** to be updated. Then **update_attributes** modifies values and saves into database if valid and permitted.

Helpers for RESTful resource paths

In the view:

```
<%= link_to "Edit Student", edit_student_path  
(@student) %>
```

write this to the HTML:

```
<a href="/students/3/edit">Edit Student</a>
```

A whole bunch more, check them out here:

<http://guides.rubyonrails.org/routing.html#path-and-url-helpers>

Custom actions and Nested Resources

What if you want a different URL to show up on your page? Modify routes.rb

```
resources :users do
  post :custom_action, on: :collection
  #POST: /users/custom_action/

  post :custom_action, on: :member
  #POST: /users/:id/custom_action/

  resources :comments, only: [:new, :create]
  #Can append standard RESTFUL paths for
  comments i.e. /users/:id/comments/new etc.
end
```


Validations

Review: Validation

```
class Student < ActiveRecord::Base
  validates :name, presence: true
  def validate_name
    if (name == "Kurt Cobain")
      errors.add(:name, "Liar!")
    end
  end
end

validate :validate_name
validates_format_of :login,
  with: /[a-zA-Z0-9_-]+/,
  message: "only letters, numbers, underscores
( ), and dashes (-) are allowed"

end
```

Custom validation method

Saves error info


Built-in validator

http://edgeguides.rubyonrails.org/active_record_validations.html

Review: Error Message Helper

```
<% @student.errors.full_messages.each do |msg| %>
  <p><%= msg %></p>
<% end %>
```

```
<% form_for(@student) do |form| %>
  ...
  <%= form.text_field(:name) %>
  ...
<% end %>
```



If there is an error, the problematic form field will be highlighted by being surrounded with **<div class="field_with_errors"></div>** automatically by Rails so you can add styling.

Valid Method (valid?)

```
class Person < ActiveRecord::Base
  validates :name, presence: true
end
```

```
Person.create(name: "John Doe").valid?
# => true

same applies to .update_attributes and .save

Person.create(name: nil).valid?
# => false
```

Review: File Uploads with Rails

```
<% form_for(@student) do |form| %>
  ...
  <%= form.file_field(:photo) %>
  ...
<% end %>
```

In form post method:

```
params[:student][:photo].read
params[:student][:photo].original_filename
```

Virtual Attributes

What if you want Model attributes that are not database columns? Define *virtual attributes* (e.g. `full_name`) with getters/setters from existing model attributes (e.g. `first_name`, `last_name`).

```
class User < ActiveRecord::Base
  def full_name
    "#{first_name} #{last_name}"
  end

  def full_name=(name)
    first_name, last_name = name.split
  end
end
```

Sessions

`session` is a `hash` that is accessible during all requests from a particular browser. Session data is `not meant to be permanent`.

Use `reset_session` to clear this data and start a new session.

Commonly used for tracking a user (such as a logged in user) or his/her information on the website.

Session Example (Very Simple!)

```
def controller_action
  ...
  session[:foo] = bar
  ...
end
```

```
def destroy_session
  ...
  reset_session
  ...
end
```


Flash

If you need to display a message to the user in the *next* page they view, use `flash`. This is useful if you're redirecting to a new page.

Commonly used for “Welcome back <Username>” and the other messages that need to be transient.

Flash Example

Controller:

```
def logout
  flash.notice = "You have logged out"
  redirect_to url
end
```

View:

```
<% if flash.notice %>
  <div><%= flash.notice %></div>
<% end %>
```

Passwords

Password Hashing

Storing passwords in **plain text** in your database is **insecure**.

Solution Attempt: Use **password hashing** to store cryptographic hashes, instead of plain-text passwords.

Note: this is not the same as the 'hashmap'

Problem: Offline dictionary attacks!

Eg. LinkedIn in 2012 password database hacked and leaked and common passwords easily broken and discovered.

Better Solution: **Salt first**, then hash passwords (Assignment 5)

Take CS 155 and/or CS 255 to learn more about security and hashing!

Password Hashing (Ruby)

Hash a string using **Digest::SHA-2**

String to be hashed!



```
Digest::SHA2.hexdigest("hello")
```

=>

```
"2cf24dba5fb0a30e26e83b2ac5b9e29e1b  
161e5c1fa7425e73043362938b9824"
```

Filters (Bonus Material)

```
class MyController < ActionController::Base
  before_filter :require_login

  private

  def require_login
    if params[:foo] == "bar"
      redirect_to "url", notice: "message"
    end
  end
end
```

http://guides.rubyonrails.org/action_controller_overview.html#filters

Form Tag

```
<% form_tag("post_submit", method: "post") do %>
  <%= label_tag(:text_val, "Enter something:") %>
  <%= text_field_tag(:text_val) %>
  <%= submit_tag("Submit") %>
<% end %>
```

Many other tags for use in `form_tag` such as:

```
check_box_tag, email_field_tag,
file_field_tag, hidden_field_tag,
password_field_tag, text_area_tag,
radio_button_tag
```

Useful resources

- <http://api.rubyonrails.org/classes/ActionView/Helpers/FormHelper.html>
- http://guides.rubyonrails.org/form_helpers.html
- http://guides.rubyonrails.org/active_record_validations.html
- http://guides.rubyonrails.org/active_record_callbacks.html
- http://guides.rubyonrails.org/action_controller_overview.html