Rails: Views and Controllers II

Lecture 32
Recall: Rails Architecture

Model

Controller

Dispatcher

Routes

Web Server

Browser

GET /hi

app/
controllers/
course_roster_controller.rb
CourseRosterController
#wake_up

app/
views/
course_roster/
wake_up.html.erb
Wiring Views and Controllers

- A controller is just an ordinary Ruby class
  - Extends ApplicationController
  ```ruby
class CourseRosterController < ApplicationController
  end
  ```
  - Location: app/controllers/
  - Filename: course_roster_controller.rb
- Actions are methods in that class
  ```ruby
def wake_up
  ... 
  end
  ```
- A view is an HTML page (kind of) that corresponds to that action
  - Location: app/views/course_roster/
  - Filename: wake_up.html.erb
  - Has access to *instance* variables (e.g., @student) of corresponding controller!
Example: books/show.html.erb

```html
<p style="color: green"><%= notice %></p>

<%= render @book %>

<div>
  <%= link_to "Edit this book", edit_book_path(@book) %>
  |
  <%= link_to "Back to books", books_path %>

  <%= button_to "Destroy this book", @book, method: :delete %>
</div>
```
Creating a Response

- There are 3 ways a controller action can create the HTTP response:
  1. Do nothing: defaults are used
  2. Call render method
  3. Call redirect method
- The first 2 result in HTTP status 200 (OK)
  - Body of response is the HTML of the view
- The 3rd results in HTTP status 302 (temporary redirect)
- Other responses are possible too (e.g., useful for ajax)
1: Default Response

- If the action does not call render (or redirect), then render is implicitly called on corresponding view

```ruby
class BooksController < ApplicationController
  def index
    @books = Book.all
  end
end
```

- Results in call to render

`app/views/books/index.html.erb`
2: Explicitly Calling Render

- Argument: *action* whose *view* should be rendered
  
  ```ruby
  def wake_up
    render :show # or render "show"
  end
  
  def show ...
  ```

- Action (show) does *not* get executed

- Action could be from another controller
  
  ```ruby
  render 'products/show'
  ```

- Can return text (or json or xml) directly
  
  ```ruby
  render plain: "OK"
  render json: @book # calls to_json
  render xml: @book # calls to_xml
  ```

- Note: render *does not* end action, so don't call it twice ("double render error")
3: Calling Redirect

- Sends response of an HTTP redirect (3xx)
  - Default status: 302 (temporary redirect)
  - Override for permanent redirection (301)
- Consequence: client (browser) does another request, this time to the URL indicated by the redirect response
  - New request is a GET by default
- Need URL, can use named route helpers
  ```ruby
  redirect_to user_path(@user)
  redirect_to @user # calls url_for(@user)
  redirect_to users_path
  redirect_to edit_user_path(@user)
  ```
- Or :back to go back in (client’s) history
Redirect vs Render

- **Similarity**
  - Point to a different view
  - Neither ends the action
  
  ```
  render... and return # force termination
  ```

- **Difference**
  - Redirect entails 2 round-trips: request, action, response, request, action response
  - Redirect requires a URL as argument, Render requires a view (action)

- **Common usage for Redirect:** POST-Redirect-GET pattern
GET Blank Form, POST the Form

Students

Fname: Marco
Lname: Portini
Buckid: 34122039

Show this student
New student

New student

GET "a blank form"

POST /students
lname: ...etc

Create Student
Back to students
GET Blank Form, POST the Form

POST /students
lname: ...etc
GET Blank Form, POST the Form

POST /students
lname: ...etc
GET Blank Form, POST the Form

POST /students
lname: ...etc
POST-Redirect-GET Pattern

User fills out order form.

User clicks SUBMIT.

POST

3xx Redirect

Insert order into the database.

User refreshes page.

Send confirmation page.

Your order was successful.

Resubmits GET request.

GET

2xx Success
Example of POST-Redirect-GET

class BooksController < ApplicationController

  def create
    @book = Book.new(book_params)
    if @book.save
      redirect_to book_url(@book), notice: 'Book successfully created'
    else
      render :new
    end
  end

end
Example of POST-Redirect-GET

class BooksController < ApplicationController

  def create
    @book = Book.new(book_params)
    if @book.save
      redirect_to book_url(@book), notice: 'Book successfully created'
    else
      render :new
    end
  end
Flash

- A hash returned with redirect response
  - Set by controller action issuing redirect
    flash[:referral_code] = 1234
  - Common keys can be assigned in redirect
    redirect_to books_url notice: '...'
    redirect_to books_url alert: '...'

- Flash included in client’s next request

- Flash available to next action’s view!
  <p id="info"><%= flash[:warn] %></p>...
  - But: flash.now available to first view!
    flash.now[:notice] = 'no such book'
Flash: Set, Use, Clear

- **User fills out order form.**
  - **User clicks SUBMIT**
  - **POST**
  - **3xx Redirect**
  - **Insert order into the database.**

- **User refreshes page.**
  - **Resubmits GET request**
  - **GET**
  - **Send confirmation page.**

- **Your order was successful.**

- **set flash**
- **use flash (then clear)**
Using Flash in View

# display just notice message
<p id="notice"><%= notice %></p>

# display all the flash messages
<% if flash.any? %>
    <div id="banner">
        <% flash.each do |key, message| %>
            <div class="flash <%= key %>">
                <%= message %>
            </div>
        <% end %>
    </div>
<% end %>
Example of Render vs Redirect

class BooksController < ApplicationController

  def update
    @book = Book.find(params[:id])
    if @book.update(book_params)
      redirect_to book_url(@book), notice: 'Book successfully created'
    else
      render :edit
    end
  end

end
Why Is This Wrong?

class BooksController < ApplicationController

  def update
    @book = Book.find(params[:id])
    if @book.update(book_params)
      redirect_to book_url(@book), notice: 'Book successfully created'
    else
      render :edit, notice: 'Try again.'
    end
  end
class BooksController < ApplicationController

  def update
    @book = Book.find(params[:id])
    if @book.update(book_params)
      redirect_to book_url(@book),
      notice: 'Book successfully created'
    else
      flash.now[:notice] = 'Try again.'
      render :edit
    end
  end
end
Code Duplication

class BooksController < ApplicationController

def show
    @book = Book.find(params[:id])
end

def edit
    @book = Book.find(params[:id])
end

def update
    @book = Book.find(params[:id])
    ...
end
DRY, aka Single-Point-of-Control

```ruby
class BooksController < ApplicationController
  before_action :set_book,
    only %i[ show edit update destroy ]

  def show  # method is now empty!
    end

  def edit  # method is now empty!
    end

  # and other actions...

  private
    def set_book
      @book = Book.find(params[:id])
    end
end
```
Sanatizing Inputs

```ruby
def update
  if @book.update(book_params)
    redirect_to @book, notice: 'Success!'
  else
    render :edit
  end
end

private
  def set_book
    @book = Book.find(params[:id])
  end

  def book_params
  end
```
Recall Partialss

- A blob of ERb used in multiple views
- Examples
  - Static header used throughout site
  - Dynamic sidebar used in many places
- Include in a template (or layout) with:
  ```ruby
  <%= render 'menu' %>
  <%= render 'users/icon' %>
  ```
- Filename of partial has "_" prefix
  - Default location: app/views
    ```ruby
    app/views/_menu.html.erb
    ```
  - Organize into subdirectories with good names
    ```ruby
    app/views/users/_icon.html.erb
    ```
Example: views/layouts/applic...

```html
<!DOCTYPE html>
<html>
  ...
  etc
<body>
  <%= render 'layouts/header' %>
  <div class="container">
    <%= yield %>
    <%= render 'layouts/footer' %>
  </div>
</body>
</html>
```
Example: views/layouts/_footer

```html
<footer class="footer">
  <small>
    <a href="http://www.osu.edu">OSU</a>
  </small>
  <nav>
    <ul>
      <li><%= link_to "About", about_path %></li>
      <li><%= link_to "Contact", contact_path %></li>
    </ul>
  </nav>
</footer>
```
Recall: Tricks with Partials

- Content of partial can be customized with arguments in call
  - In call: pass a hash called :locals
    ```ruby
    <%= render partial: "banner",
      locals: { name: "Syllabus,
                amount: @price } %>
    ```
  - In partial: access hash with variables
    ```html
    <h3> <%= name %> </h3>
    <p> Costs <%= "$#{amount}.00" %> </p>
    ```
Parameter Passing to Partials

- Partial also has one *implicit* local variable
- In the partial, parameter name *same* as partial
  
  ```
  # in partial nav/_menu.html
  <p> The price is: <%= menu %></p>
  ```

- Argument value assigned explicitly
  ```
  <%= render partial: 'nav/menu',
      object: cost %>
  ```

- Idiom: Begin partial by renaming this parameter
  ```
  # in partial nav/_menu.html
  <% price = menu %>
  ```
Example: books/show.html.erb

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

<%= render @book %>

<div>
  <%= link_to "Edit this book", edit_book_path(@book) %>
  |
  <%= link_to "Back to books", books_path %>

  <%= button_to "Destroy this book", @book, method: :delete %>
</div>
<div id="<%= dom_id book %>">
  
  <p>
    <strong>Title:</strong> <%= book.title %>
  </p>

  <p>
    <strong>Author:</strong> <%= book.author %>
  </p>

</div>
Demo: Scaffolding

- Generate many things at once
  - Migration for table in database
  - Model for resource
  - RESTful routes
  - Controller and corresponding methods
  - Views for responses

- Command
  
  $ rails g scaffold Student lname:string buckid:integer
  $ rails db:migrate
  $ rails server
Summary

- Controller generates a response
  - Default: render corresponding view
  - Explicit: `render` some action's view
  - Explicit: `re-direct`
  - POST-redirect-GET (aka “get after post”)
  - Flash passes information to next action

- Reuse of views with partials
  - Included with render (e.g., `<%= render ...`)
  - Filename is prepended with underscore
  - Parameter passing from parent template
  - Can iterate over partial by iterating over a collection
Partials With Collections

- Iteration over partials is common
  ```
  <%= for item in @items %>
    <%= render partial: 'item_brief', object: item %>
  <% end %>
  ```
- Short-hand: Replace above with
  ```
  <%= render partial: 'item_brief', collection: @items %>
  ```
- Renders partial once for each element
- Initializes partial local variables each time
  - `item_brief` (the member of the collection)
  - `item_brief_counter` (integer 0..size of collection)
- Can also add separator `between` each partial
  ```
  <%= render partial: 'item_brief', collection: @items, spacer_template: 'line_space' %>
  ```
Partial Super Shorthands

- For a model *instance* (e.g. `@book`) in a template
  
  ```erb
  <%= render @book %>
  ```
  
  - Includes `_book.html.erb` partial
  - Passes in `@book` to partial (as `:object`)
  - Value available as local variable `book` in partial

- For a model *collection* (e.g. `@books`) in a template
  
  ```erb
  <%= render @books %>
  ```
  
  - Call render multiple times, once/member
  - Each call uses same partial (`_book.html.erb`)
  - Each call passes in different member as argument
  - Value available as local variable `book` in partial

- Returns nil if collection is empty
  
  ```erb
  <%= render @books || 'No books to see.' %>
  ```