Rails:
Views and Controllers II

Lecture 29
Recall: Rails Architecture

app/controllers/course_roster_controller.rb
CourseRosterController
#wake_up

GET /sing

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
```
- 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 (eg @student) of corresponding controller!
Layout

- Returned HTML formed from: **Layout + template**
- Layout is the overall structure of the HTML page
- See: `app/views/layouts/application.html.erb`

```html
<!DOCTYPE html>
<html>
<head>
  
  <title> ... etc
  
</head>
<body>

  <%= yield %>

</body>
</html>
```

- Action's template replaces layout's yield
- Template is action-specific content of response
- Layout is where you put site-wide styling
  - Eg navigation bar, div's with CSS classes
Example: books/index.html.erb

<h1>Listing Books</h1>
<table>
  <tr>
    <th>Title</th> <th>Summary</th> <th></th> <th></th> <th></th>
  </tr>
  <% @books.each do |book| %>
  <tr>
    <td><%= book.title %></td>
    <td><%= book.content %></td>
    <td><%= link_to 'Show', book %></td>
    <td><%= link_to 'Edit', edit_book_path(book) %></td>
    <td><%= link_to 'Remove', book, :confirm => 'Are you sure?', :method => :delete %></td>
  </tr>
  <% end %></table>

<br />
<%= link_to 'New book', new_book_path %>
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 (eg useful for ajax)
1: Default Response

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

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

- Results in call to render of

  `app/views/books/index.html.erb`
2: 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 :text => "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
- Consequence: client browser does another request, to the URL indicated in the redirect response
  - New request is a GET by default
- Default status: 302 (temporary redirect)
  - Override for permanent redirection (301)
- Same arguments as `link_to`
- Or `:back` argument to go back in history
- Note: Redirect vs Render
  - Feel similar in pointing to a different location
  - But very different semantics
- Common usage: POST-Redirect-GET pattern
GET Blank Form, POST the Form

GET "a blank 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.

Send confirmation page.

Your order was successful.

User refreshes page.

2xx Success

GET

Resubmits GET request.
Example of Render vs Redirect

class BooksController < ApplicationRecord

  def create
    @book = Book.new(book_params)
    if @book.save
      redirect_to :action => :show
    else
      render :new
    end
end
Example of Render vs Redirect

class BooksController < ApplicationController

  def update
    @book = Book.find(params[:id])
    if @book.update_attributes(params[:book])
      redirect_to :action => 'index'
    else
      render :edit
    end
  end
Partials

- 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:
  ```erb
  <%= render :partial => 'menu' %>
  <%= render 'users/icon' %>
  ```
- Filename of partial has "_" prefix
  - Default location: app/views
    ```erb
    app/views/_menu.html.erb
    ```
  - Organize into subdirectories with good names
    ```erb
    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>
```
Parameter Passing to Partialss

- Partial's implicit *local* variable
  - Argument value assigned explicitly
    ```ruby
    <%= render :partial => 'shared/menu',
            :object => cost %>
    ```
  - In the partial, variable name *same* as partial
    ```html
    <p>The price is: <%= menu %></p>
    ```
  - Idiom: Begin partial by renaming this variable
    ```html
    <%= price = menu %> 
    ```

- Short-hand, when partial name matches:
  ```ruby
  <%= render :partial => @book %>
  ```
  ```html
  The <em><%= book.title %></em> is...
  ```

- Alternative: explicit naming of locals
  ```ruby
  <%= render :partial => 'shared/menu',
            :locals => { :price => 35 } %>
  ```
Example: view/books/index

<h1>Listing Books</h1>
<table>
<tr>
  <th>Title</th> <th>Summary</th> <th></th>
</tr>

<% @books.each do |book| %>
<tr>
  <td><%= book.title %></td><td><%= book.content %></td><td><%= link_to 'Show', book %></td>
  <td><%= link_to 'Edit', edit_book_path(book) %></td>
  <td><%= link_to 'Remove', book, :confirm => 'Are you sure?', :method => :delete %></td>
</tr>
<% end %></table>

<br /><%= link_to 'New book', new_book_path %>
<h1>Listing Books</h1>
<table>
  <tr>
    <th>Title</th> <th>Summary</th>
  </tr>
  <% @books.each do |book| %>
    <%= render :partial => 'detail', :object => book %>
  <% end %>
</table>
<br />
<%= link_to 'New book', new_book_path %>
## Corresponding Partial

```
<tr>
  <td><%= detail.title %></td>
  <td><%= detail.content %></td>
  <td><%= link_to 'Show', detail %></td>
  <td><%= link_to 'Edit', edit_book_path(detail) %></td>
  <td><%= link_to 'Remove', detail, :confirm => 'Are you sure?', :method => :delete %></td>
</tr>
```
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* (eg @book) in a template

```ruby
<%= render @book %>
```
- Includes _book.html.erb partial
- Passes in @book to partial (available as local variable book within partial)

For a model *collection* (eg @books) in a template

```ruby
<%= render @books %>
```
- Call render multiple times, once/member
- Each call uses same partial (_book.html.erb)
- Each call passes in different member as argument (local variable book in partial)

Returns nil if collection is empty

```ruby
<%= render @books || 'No books to see.' %>
```
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`  
  - `$ rake db:migrate` 
  - `$ rails server`
Summary

- Controller generates a response
  - Default: render corresponding view
  - Explicit: call to some action's render
  - Explicit: redirect
  - POST-redirect-GET (or "get after post")
- Reuse of views with partials
  - Included with render (eg `<%= render...`)
  - Filename is prepended with underscore
  - Parameter passing from parent template
  - Can iterate over partial by iterating over a collection