Rails: Views and Controllers

Lecture 18
Recall: Rails Architecture
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
  ```ruby
  Location: app/views/course_roster/
  Filename: wake_up.html.erb
  Has access to instance variables (eg @student) of corresponding controller!
  ```
Recall: Rails Architecture

app/controllers/course_roster_controller.rb
CourseRosterController
#wake_up

app/views/course_roster/wake_up.html.erb

GET /sing
$ rails new demoapp

Create CourseRosterController

```ruby
class CourseRosterController < ApplicationController
end
```

Create (empty) method wake_up

Add route to config/routes.rb

```ruby
get 'sing' => 'course_roster#wake_up'
```

Create wake_up.html.erb

- In app/views/course_roster

$ rails server
Example: Controller

```ruby
# in app/controllers/
# filename course_roster_controller.rb

class CourseRosterController < ApplicationController
  def wake_up
    # for this simple example, no code needed
  end
end
```
Example: Route Definition

```ruby
#in config/
#filename routes.rb

Demoapp::Application.routes.draw do
  get 'sing' => 'course_roster#wake_up'
#equivalent to (but shorter than):
#  match 'sing' => 'course_roster#wake_up',
#       :via => [:get]
end
```
Example: View

<!-- in app/views/course_roster/
      filename wake_up.html.erb -->

<h1>Yo!!</h1>

<p>Are you awake?</p>
Single Point of Control

- Notice the duplication in names
  - Controller name (*course_roster*) used in:
    - Name of the controller class
    - Filename containing the controller class
    - Route
    - View's parent directory
  - Action name (*wake_up*) used in:
    - Name of the method within controller class
    - Route
    - Filename containing the view
- Solution: generate all these parts
  ```bash
  $ rails g controller CourseRoster wake_up
  ```
$ rails generate controller Prof ask_question visit_office

Results in:
- Modification to config/routes.rb
- Creation of ProfController class
  app/controllers/prof_controller.rb
- Definition of methods in ProfController
  def ask_question ... end
def visit_office ... end
- Creation of 2 views (ie one per action)
  app/views/prof/ask_question.html.erb
  app/views/prof/visit_office.html.erb

$ rails server
Recall ERb: Embedded Ruby

- General templating mechanism
  - A string (usually contents of a file, "template")
  - Escaped bits of ruby
    - `<% code %>` execute ruby code ("scriplet")
    - `<%= expr %>` replaced by result of expr
    - `<%# text %>` is a comment

- Example: a text file
  This is some text.
  `<% 5.times do %>`
  Current Time is `<%= Time.now %>`!
  `<% end %>`

- Naming convention: filename.outputlang.erb
  - Eg `index.html.erb`

- Popular alternative: HAML
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 %>
Recall 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
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 class `BooksController < ApplicationController`

  ```ruby
  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 button.
POST
Redirect to inser order into the database.
Redirect again to get request.
GET
Send confirmation page.
User refreshes page.
Your order was successful.
Example of Render vs Redirect

class BooksController < ApplicationController

  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

<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 Partials

- 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
    ```ruby
    <%= 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

```html
<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 %>
```
<h1>Listing Books</h1>
<table>
  <tr>
    <th>Title</th> <th>Summary</th> <th></th> <th></th> <th></th>
  </tr>
  <% @books.each do |book| %>
    <%= render :partial => 'detail', :object => book %>
  <% end %>
</table>

<br />
<%= link_to 'New book', new_book_path %>
<table>
<thead>
<tr>
<th>Title</th>
<th>Content</th>
<th>Show Link</th>
<th>Edit Link</th>
<th>Remove Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>detail.title</td>
<td>detail.content</td>
<td>detail</td>
<td>edit_book_path(detail)</td>
<td>link_to('Remove', detail, :confirm =&gt; 'Are you sure?', :method =&gt; :delete)</td>
</tr>
</tbody>
</table>

</tr>
Partials With Collections

- **Iteration over partials is common**
  ```erb
  <% for item in @items %>
    <%= render :partial => 'item_brief',
              :object => item %>
  <% end %>
  ```

- **Short-hand: Replace above with**
  ```erb
  <%= 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**
  ```erb
  <%= render :partial => 'item_brief',
             :collection => @items,
             :spacer_template => 'line_space' %>
  ```
Partial Super Shorthands

- For a model instance (eg @book) in a template
  ```erb
  <%= 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
  ```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 (local variable book in partial)

- Returns nil if collection is empty
  ```erb
  <%= 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