Week 4 Lecture 12

Applied

Helpful Resources

- http://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/
- http://guides.rubyonrails.org/layouts_and_rendering.html

What's in this lecture?

- The 'V' in MVC: Views!
- How views work in Ruby on Rails
- ERB Templates; Mixing code with HTML

Understanding the View

- In the pure sense: GUI for that model
- In the practical: GUI for app function
- Typically HTML, but can be XML, JSON, ...

Chef Analogy

- A restaurant has many dishes to serve:
  - MODEL = Menu item
  - VIEW = Plate
  - CONTROLLER = Chef
- A patron orders a dish (model), and the chef (controller) prepares the meal on the plate (view)

Rails MVC

app/controllers/people_controller.rb
../posts_controller.rb
app/models/person.rb
../post.rb
app/views/people/index.html.erb
../edit.html.erb
../_form.html.erb
../posts/index.html.erb
../edit.html.erb
../_form.html.erb

General Form MVC

app/controllers/model_name_controller.rb
app/models/model_name.rb
app/views/model_name/action_01.html.erb
../action_02.html.erb
../action_03.html.erb
../action_N.html.erb

Understand: Models

- Model is responsible for the integrity of the data
  - different models for different dishes
  - the dish itself

Understand: Controllers

- controller is the gatekeeper for the model
  - chef is responsible for access to different components: appetizer, salad, main, desert
  - forwards ‘data’ to ‘view’
  - Assembly and coordination

Hold Up

- There is always design tension between the controllers and models -- remember: it is an art!

Understand: Views I

- view is the multiple presentations of the model
- chef analogy: for here view; to go view
- presentation and serving

Understand: Views II

- we can create multiple:
  - views of the data
    - XML, JSON, HTML
  - ways of accessing data
    - API
    - Web view

Rails Views

- How do our requests even get to views?
  - Routing -> maps request to controller#action
  - Processing -> controller#action builds instance variables required in view
  - View rendering -> controller#action maps to a view template and processed as Ruby code
  - Server responds with rendered view code!

Templates

- Allow us to mix HTML with Ruby
- Why would we want to do that?
  - Recycling and reusing view code
  - Cleaner implementation for maintaining
  - Single change propagates through app

In practice:

```
<% @people.each do |person| %>
  <div id="<%= person.id %>">
    <p><%= person.name %></p>
  </div>
<% end %>
```

Views

- Ruby code needs to be written between:

  <% name = @model.name %>

- When the **value** needs to be output:

  <%= @model.name %>

Why?

- This allows us to use the powerful looping constructs from Ruby to generate HTML
- Allows you to build views with repeating patterns and structures, and reuse code
- Instance variables and data are now accessible to be rendered to the view

Hint:

- All data needed by the view should be set up in the controller.
- i.e. do not add to view:

  <%= ModelName.find(:first).name %>

- Instead add to controller:

  @name = ModelName.first

- And the view:

  <%= @name %>

Using view helpers

- View Helpers are Ruby methods that generate common HTML elements
- Part of the ActionView API
- Things like forms, include tags, and buttons can all be written as Ruby code

Example

```
<% form_for @post do |f| %>
  <%= f.text_field = @post.name %>
  <%= f.submit %>
<% end %>
```

Exercises

- Create a view that generates HTML for all elements in an array
- Create a view that generates HTML for all key-value pairs in a hash
- Create a form using Rails' view helpers