Controllers and Views

COSC 480
23 February 2018

Joel Sommers
jsommers@colgate.edu
Colgate University

Adding a new action to a Rails app

1. Create a route in `config/routes.rb`, if needed
2. Add the action (method) in the appropriate controller
   - `app/controllers/*_controller.rb`
3. Ensure there is something for the action to render
   - `app/views/*model*/*action*.html.haml`
   - *e.g.*, `app/views/products/show.html.haml`

Working example (with some review)

- Create a Product model with name (string) and price (decimal 8.2)
- Create a db seed file to initially populate the db
- Create RESTful routes for Product
- Create index and show controller methods and corresponding view templates
  - index: retrieve and display all products
  - show: show details for one product

Steps for doing "index" route (controller + view)

- Use the rails generators to create controller and view template
  - Don't forget to add `haml-rails` to Gemfile
  - `rails generate controller Products`
  - Can also say `rails generate controller Products index show` to get stubbed-out view files
  - Edit `app/controllers/products_controller.rb`
    - Retrieve all objects from products table; make available to view
    - That's it!
  - Edit `app/views/products/index.html.haml`
  - Application template render starts with *layout* page, which yields to individual page templates
    - `app/views/layouts/application.html.haml` (or `.erb`)
    - Can set common elements on layout, such as CSS includes, "meta" tags, common page titles, etc.

Rails naming conventions

- Model class name is singular `UpperCamelCase`, e.g., `Product`, `RentalProperty`
- DB table is snake-cased, and plural, e.g., `products`, `rental_properties`
- Controller class is `UpperCamelCase` and plural, e.g., `ProductsController`
- Controller methods are lower snake case, e.g., `index`, `sale_items`
- View templates are named after individual controller methods, e.g., `app/views/products/index.html.haml`

Views: Haml and CSS

- Haml reference: `http://haml.info`
  - Initial focus should be on page *structure*, not styling
  - Indentation matters in Haml! (And it's even more picky than Python)
  - Tags start with `%` — Haml automatically closes tags
  - Use `-` to start Ruby code that shouldn't have output on the page
  - Use `=` to start Ruby code that should have output displayed on page
  - Any `do` blocks don't need an `end` in Haml — end is inferred by indentation
- Once you've got structure set, add `id` and `class` selectors for styling structurally and logically related elements
  - Nice CSS reference: `https://developer.mozilla.org/en-US/docs/Web/CSS/Reference`
  - Useful color scheme site: `http://colorschemedesigner.com` (or `http://paletton.com`)
  - Useful webfonts: `https://www.google.com/fonts/`
  - Useful CSS library: `http://getbootstrap.com`

Example controller method and index template

Controller: (`app/controllers/products_controller.rb`)

```
class ProductsController < ApplicationController
  def index
    @products = Product.all
  end
end
```

View template (w/o styling): (`app/views/products/index.html.haml`)

```
%table
  %caption All products
  %tr
    %th Name
    %th Price
  - @products.each do |p|
    %tr
      %td= p.name
      %td= p.price
```

MVC revisited, and the show method

- Model: methods to get/manipulate data
- Core business logic is contained in Model methods

```
Product.where(...)
Product.find(...)
```

- Controller: get data from model, make available to view

```
def show
  @product = Product.find(params[:id])
  # instance variables set in controller, are
  #          available in the view
  # absent other info, Rails will automatically look
  #          to render app/views/products/show.html.haml
end
```

- View: display/render data, allow user interaction
  - Show details of a product (name, price)
- But ...
  - What else can user do from this page?
  - How does a user get to this page?

Getting from here to there: URI helpers

Helper method	URI returned	RESTful Route and action	
products_path	/products	GET /products	index
products_path	/products	POST /products	create
new_product_path	/products/new	GET /products/new	new
edit_product_path(m)	/products/1/edit	GET /products/:id/edit	edit
product_path(m)	/products/1	GET /products/:id	show
product_path(m)	/products/1	PUT /products/:id	update
product_path(m)	/products/1	DELETE /products/:id	destroy

- Left-hand column of `rails routes` gives name of helper except for `_path` suffix
- `link_to product_path(3)` in view template generates correct URL to trigger show action
- Clicking on link hits correct route

```
GET /products/:id
    {:action=>"show", :controller=>"products"}
params[:id] == 3
```

- We get into controller `show` action with correct id, retrieve object from database, and render the `show` view

Checkpoint

What is true about Rails URIs and routes?

1. A route consists of both a URI and an HTTP method
2. A route URI must be generated by URI helper methods
3. A route URI can be generated by URI helper methods, but doesn't strictly need to be

A. Only (1) is true
B. Only (3) is true
C. (1) and (3) are true
D. (1) and (2) are true

Checkpoint

What is true about Rails URIs and routes?

1. A route consists of both a URI and an HTTP method
2. A route URI must be generated by URI helper methods
3. A route URI can be generated by URI helper methods, but doesn't strictly need to be

A. Only (1) is true
B. Only (3) is true
C. (1) and (3) are true
D. (1) and (2) are true

C

What else can we do?

- How about letting user return to products list?
- RESTful URI helper to the rescue again
  - `products_path` with no args links to Index action

    `=link_to 'Back to Products list', products_path`
- Interestingly, some HTTP methods are not actually supported by (most) browsers (e.g., PUT and DELETE)
  - This deficiency is handled transparently by Rails
  - Take a look at generated HTML if you're curious
- Example app: add links from index to show (details) pages on index template
- Add controller method for show
- Add view for show, along with link back to index

Checkpoint

Why must every interaction with a Rails SaaS app eventually cause something to be rendered?

A. Because of convention over configuration

B. Because HTTP is a request/reply protocol

C. Because Model-View-Controller implies that every action renders its own view

D. Because there must be a return value from controller methods

Checkpoint

Why must every interaction with a Rails SaaS app eventually cause something to be rendered?

A. Because of convention over configuration

B. Because HTTP is a request/reply protocol

C. Because Model-View-Controller implies that every action renders its own view

D. Because there must be a return value from controller methods

B. It's all because we're relying on HTTP!