Rails and routing

INFO 2310:
Topics in Web Design and Programming

Reminder

No class next Friday 9/19

We're back 9/26

Today's topics

- Model validation
  - How can we make sure people enter stuff we want?
- More about views and controllers
  - How does Rails decide what pages to show?
  - RESTful design
- Embedded Ruby (erb)
  - How can we use Ruby in our HTML?

Model validation

Remember the CD catalog from INFO 230? *You can't trust user input*. So how do we deal with that in Rails?

Model validation

It turns out to be particularly simple in Rails.

Open up your Post model file (blog/app/models/post.rb). Add the line:

```
validates_presence_of :title, :body
```

to the Post model.

Now try it…

Fire up the webserver (`ruby script/server` from within your blog directory) and open up a browser (to `http://localhost:3000/posts`).

Try entering/editing a post to have a blank title and/or body.

We can be slightly more sophisticated. Add another line to the Post model:

```
validates_format_of :title, :with =>
/^[\w\d]+$/
```

Now try to see what happens…

Lots of possibilities…

validates_uniqueness_of
validates_numericality_of
validate_on_create :methodname
validate_on_update :methodname
…

Errors?

How are the errors getting displayed?

Each ActiveRecord object has a list of errors (e.g. @post.errors).

If you look at
blog/app/views/post/new.erb.html
blog/app/views/post/edit.erb.html
you'll see a method that prints out the errors in this list:
`<%= f.error_messages %>`

More about views and controllers

From last time: MVC

Recall from last time: Rails uses the MVC (model-view-controller) pattern of software architecture
- Model: Objects holding data + methods for operating on them.
- Controllers: Takes requests from user interface and decide which views to render.
- Views: The HTML + Ruby that displays data from the model, gets input from the user.

From last time: Models

In working on our blog, we created a model ‘Posts’ with titles and bodies. We saw how we could manipulate data in the model.

This time: Views and controllers

How does Rails take a URL and decide what to show you?

Routes.rb

Everything starts in the routes.rb file.

Open up blog/config/routes.rb.

Routes.rb

```
ActionController::Routing::Routes.draw do
  |map|
  map.resources :posts
  map.connect ':controller/:action/:id'
  map.connect
      ':controller/:action/:id.:format’
end
```

Figuring a route

Each map.something command designed to take a URL, parse it, and direct it to the appropriate controller and method (action).

Route is decided on by first matching URL in routes.rb.

```
E.g. For our current mapping,
        /users/show/1 would match
                map.connect ‘:controller/:action/:id’
        with  params = {   :controller => “users”,
                           :action => “show”,
                           :id => 1  }
```

Would call on users_controller.rb and look for ‘show’ method. (if we had a users_controller). ‘show’ can access params[:id].

Let’s add some routes

First, let’s add a route for the root, so we don’t get the default Rails screen.

Add
```
map.root :controller => ‘posts’, :action
  => ‘index’
```
to routes.rb, just before the map.connect :controller/:action/:index line.

Also delete blog/public/index.html (or rename it).

Try it!

Another route

Let's allow us to look up blog posts by date.

As the first routing line in routes.rb (after ActionController…), add

```
map.connect 'posts/bydate/:year/:month/:day',
     :controller => "posts",
     :action => "show_date",
     :requirements => { :year => /(19|20)\d\d/,
                        :month => /[01]?\d/,
                        :day => /[0-3]?\d/ },
     :month => nil,
     :day => nil
```

(Note: for reasons we'll discuss in a minute, this isn't something we would really want to do given how posts current works).

Adding an action to a controller

Now open up blog/app/controller/posts_controller and add the following method at the bottom (just before the 'end').

```
def show_date
  @posts = Post.find(:all)
  @posts = @posts.select {|x| x.created_at.year
  == params[:year].to_i}
  @posts = @posts.select {|x| x.created_at.month
  == params[:month].to_i} if params[:month]
  @posts = @posts.select {|x| x.created_at.day ==
  params[:day].to_i} if params[:day]
  render(:action => :index)
end
```

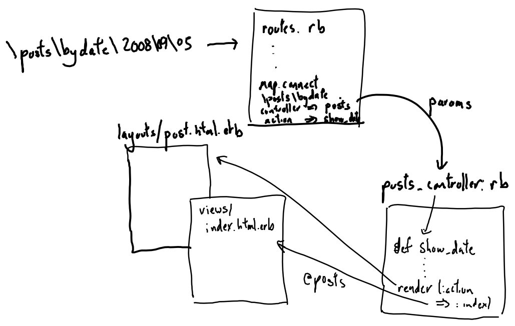
Try it!

Try entering corresponding URLs into the browser.

http://localhost:3000/posts/bydate/2008

http://localhost:3000/posts/bydate/2008/09/05

What is happening?



Views

Each controller/action *may* have an associated layout/view.

The 'posts' controller has an associated layout in app/layouts/posts.html.erb.

The views associated with the actions of the 'posts' controllers are in app/views/post/… (ones for 'index', 'edit', 'new', and 'show').

Views

When an action is called, the corresponding view is rendered (unless another render or a 'redirect' is called).

The view is output within a layout; posts.html.erb in this case.

If the corresponding layout does not exist, application.html.erb is used instead (useful if you want one layout for many controllers).

In our case…

‘show_date’ asks to render ‘index’. So app/views/posts/index.html.erb is rendered in the context of the layout app/layouts/posts.html.erb (with the @posts variable set as given in ‘show_date’).

Posts layout

```
!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="content-type" content="text/html;charset=UTF-8" />
  <title>My Blog <%= @title %></title>
  <%= stylesheet_link_tag 'scaffold' %>
</head>
<body>
  <p style="color: green"><%= flash[:notice] %></p>

  <%= yield  %>

  </body>
</html>
```

Useful tricks

Since layouts are evaluated *after* the view, they can use variables set in the view.

Try this:
in app/views/posts/show.html.erb add the line

```
<% @title = ": " + @post.title %>
```

somewhere.

Then in app/views/layouts/post.html.erb, change the <title> tag to

```
<title>My Blog <%=h @title
  %></title>
```

Try it!

REST

But in fact, the only route related to posts in routes.rb was

```
map.resources :posts
```

How does this manage to do everything that it does?

REST

REST = Representational State Transfer

Basic ideas:

- All interactions between client and server handled by a small number of ‘verbs’ applied to a larger number of well-defined ‘nouns’ or ‘resources’. ‘Resources’ can have multiple ‘representations’. Long-term state maintained by the ‘resources’.
- In our case:
  - ‘verbs’ are HTTP methods (GET, POST, PUT, DELETE)
  - ‘nouns’ are URLs (e.g. /posts/1).
  - ‘representations’ are formats (HTML, XML, RSS, JSON, etc.)

REST cont.

Why is this useful?

- Useful for networking components to know when they can cache responses.
- Rather than using the URL to indicate the action (e.g. ‘/posts/get_article/1’), have standard action (HTTP GET) applied to a resource (e.g. ‘posts/1’).
- Generalizes to other resources (e.g. we know what happes if we do an HTTP GET for ‘/users/1’).
- But at some level, I don’t get the fuss.

REST in Rails

Rails is set up for ‘RESTful’ applications.

Can see the routes created by “map.resources :posts” by typing `‘rake routes’.`

HTTP method	URL	Action	
GET	/posts	index	Lists all posts
GET	/posts/:id	show	Show post :id
GET	/posts/:id/edit	edit	Edit post :id
GET	/posts/new	new	Make new post (form input)
PUT	/posts/:id	update	Update post :id using info from request
DELETE	/posts/:id	destroy	Delete post :id
POST	/posts	create	Make new post using info from request

app/controllers/

posts_controller.rb

posts.xml

We can see the actions in the controller:

```
 def index
    @posts = Post.find(:all)
    respond_to do |format|
         format.html # index.html.erb
         format.xml  { render :xml => @posts }
    end
end

def show
    @post = Post.find(params[:id])
    respond_to do |format|
         format.html # show.html.erb
         format.xml  { render :xml => @post }
    end
end
```

XML

Note that there is built-in support for an XML representation; try browsing ‘`http://localhost:3000/posts.xml`’.

ERB

Now some of the .erb files make more sense. index.html.erb:

```
<% for post in @posts %>
<tr>
<td><%=h post.title %></td>
<td><%=h post.body %></td>
<td><%= link_to 'Show', post %></td>
<td><%= link_to 'Edit', edit_post_path(post) %></td>
<td><%= link_to 'Destroy', post, :confirm => 'Are you sure?', :method =>
   :delete %></td>  </tr>
<% end %></table>
<br />

<%= link_to 'New post', new_post_path %>
```

executed

executed and displayed

links to 'show' action

links to edit action

link to 'destroy' action

'new' action

ERB

Any Ruby inside “<% … %>” gets executed.
  E.g. `<% for post in @posts %>`

Any Ruby inside “<%= … %>” gets executed,
  the result turned into a string, and displayed.
  E.g. `<%= h post.title %>`
  ‘h’ is a method that displays special characters correctly in HTML; like PHP htmlentities().

ERB

‘link_to’ a method for creating links.

edit_post_path(post), new_post_path methods automatically created to return URLs to the ‘edit’ and ‘new’ actions of the posts_controller.

Note in ‘Destroy’ link we have to specify the HTTP method ‘:delete’.

```
<%= link_to 'Destroy', post, :confirm =>
  'Are you sure?', :method => :delete %>
```

Reminder

No class next Friday 9/19

We’re back 9/26

What happens when we add another model? How can we link two models?