6.170 Tutorial 2 - Rails Basics

Prerequisites

1. RoR Installed and "Hello World" app from P0 working.
2. Familiarity with "Separation of Concerns" and M-V-C Controller.
3. Some knowledge of databases is useful but not necessary.

Goals of this Tutorial

Get familiar with the some basics of development using Ruby on Rails.

Tutorial

Note: Read the README.rdoc file of your app. It has some very useful information for developing and debugging Rails apps. You'll also find a lot of useful guides from http://guides.rubyonrails.org/index.html.

You Must Have Already Figured:

1. What is Ruby on Rails?
    a. A framework for rapid web-based development using the Ruby language
    b. A community of passionate developers and contributors.
2. What features does it provide?
    a. Clean Separation of concerns: Model-View-Controller.
    b. Rapid web-based software development and testing.
    c. Rapid DB Access and Migrations - without knowing many details of databases.
    d. rake, a "make" system for Ruby for automating repetitive tasks.
    e. Excellent software testing tools - details will be covered in a future recitation.
    f. A vibrant community of developers and contributors who make life easy for you.
3. What is Heroku? What features does it provide?
    a. A cool hosting service for RoR apps - makes deploying and running complete RoR applications very easy.
    b. Is based on Amazon EC2 infrastructure; allows dynamic provisioning of web servers, and backend servers.
4. What is Git? GitHub?
    a. Git is an awesome source control system and GitHub is an online service for easy creation and sharing of Git repositories.

Topic #0: Useful commands when developing rails applications

"`rails`" command provides many useful scripts for auto code generation etc. Examples:

```
$ rails [command] [args]
$ rails new <project>
$ rails generate <asset | model | resource etc.> <parameters>
```

```
$ rails destroy <asset | model | resource etc.> <parameters>
```

"rake" is Ruby's make system. Many useful development tasks come included as rake tasks
Examples:

```
$ rake -T
$ rake db:setup; rake db:migrate; rake db:drop:all
$ rake test; rake test:uncommitted
```

Note: Many common tasks come built-in with rails but you can always develop custom rake tasks to automate your own work.

Topic #1: Routing in Rails

The material in this section is adapted from: http://guides.rubyonrails.org/routing.html
Also see documentation in your `routes.rb` file

Rails Router

1. Maps a URI to methods in controllers
    a. Example: `post "home/login" => "home#login"`
2. Utility methods for generating paths and URLs - no hard-coding of URLs
3. Routing specified in config/routes.rb config file

Two kinds of Routing in Rails

1. Resourceful Routing: Shorthand for resources
2. Non-Resourceful Routing: For all other routing

Resourceful Routing Examples

At the command line in your app directory:

```
$ rails generate resource book title:string author:string
price:number
```

Check `config/routes.rb` file for:

```
resources :books
```

This creates seven different routes in your application, all mapping to the Books controller:

HTTP Verb	Path	action	used for
GET	/books	index	display a list of all books
GET	/books/new	new	return an HTML form for creating a new books

POST	/books	create	create a new book
GET	/books/:id	show	display a specific book
GET	/books/:id/edit	edit	return an HTML form for editing a book
PUT	/books/:id	update	update a specific book
DELETE	/books/:id	destroy	delete a specific book

In your browser, go to http://localhost:3000/books/new You will see an error like :

Unknown action

The action 'new' could not be found for BooksController

Now edit the `app/controllers/book_controller.rb` file so that it looks like:

```
class BooksController < ApplicationController

def index
 render "index", :layout => false
end

def new
 @time_now = Time.now
 ## render "new", :layout => false
end

end
```

Add file `app/views/books/index.html.erb` so that it looks like:

```
<html>
 <head>
 </head>
 <body>
   <h1> Show a listing of all books here </h1>
```

```
    <%= "This is dynamically generated content at: #{Time.now}" %>
  </body>
</html>
```

Add file `app/views/books/new.html.erb` so that it looks like:

```
<html>
  <head>
  </head>
  <body>
    <h1> Show form for creating a new book here </h1>
    <%= "This is dynamically generated content at: #{@time_now}" %>
  </body>
</html>
```

Now http://localhost:3000/books and http://localhost:3000/books/new will work. Similarly, other action methods like `show update, edit destroy` can be defined on Books controller and corresponding view files written.

Notice in this example:

1. **:render** directive and its parameters
2. Rendering of appropriate view even when **:render** is not specified in `"new"` method
3. Dynamic content in view files and variable scope + visibility between controllers and views

Examples of Non-Resourceful Routing

```
## Static routes
post /home/login => "home#login"
post /library/catalog/update => "library/catalog#update"
get /home/index

## Examples of routes with dynamic segments
match ":controller/" => ":controller#index"
```

http://localhost:3000/books will map to `BooksController.index`

```
match ":controller/:action"
```

http://localhost:3000/books/deleteall will map to `BooksController.deleteall`
Warning: This route will match every method in every controller. Use cautiously.

```
match ":controller/:action/:user_id"
```

http://localhost:3000/books/checkout/10 will map to `BooksController.checkout` with params[:user_id] = 10

Note: You can mix static and dynamic segments in the match directive.

Helper Methods for Routing in Controllers and Views
resource_path, resource_url. For resource **:books**

- books_path returns /books
- new_book_path returns /books/new
- edit_book_path(:id) returns /books/:id/edit (for instance, edit_book_path(10) returns /books/10/edit)
- book_path(:id) returns /books/:id (for instance, book_path(10) returns /books/10)

Each of these helpers has a corresponding _url helper (such as books_url) which returns the same path prefixed with the current host, port and path prefix.

Inspecting and Testing Routes

```
$ rake routes
```

More Routing Concepts
Namespaces in Resourceful Routing
You can define namespaces for each of your resources as shown below:

```
namespace :admin do
 resources :posts, :comments
end
```

Nested Resources
See **http://guides.rubyonrails.org/routing.html#nested-resources**

URL Redirection Using Routes
See **http://guides.rubyonrails.org/routing.html**

Route Globbing

```
match 'books/*other' => 'books#unknown'
```

This route would match “books/12” or “/books/long/path/to/12”, setting params[:other] to "12" or "long/path/to/12".

Topic #2: Views and Client Side Assets
Also see: http://guides.rubyonrails.org/layouts_and_rendering.html

We will use “erb” format for dynamic views. You are welcome to use and experiment with other

view technologies like HAML and choose the one which suits you best. But not all TAs may not be able to help you if you encounter problems - only one of the TAs uses HAML regularly.

View Basics
Dynamic content specified in erb files via "<% %>" delimiters

```
<% [some_ruby_code] %>
<%= [some_ruby_code_which_will_be_printed] %>
```

Note: Only Instance scoped variables from controller available to views

Examples:
Store the current time in "now":

```
<% now = Time.now %>
```

Output the current time:

```
<%= Time.now %>
```

Print a HTML table of all books

```
<table>
<% @books.each.do |book| %>
<tr>
      <td><%= book.title%></td>
      <td><%= book.author%></td>
      <td><%= book.price%></td>
</tr>
<% end %>
</table>
```

Note: @books must be previously initialized - preferably in the controller. Otherwise it will be nil and you will get an error

Output a HTML form for entering book information:

```
<%= form_for @book do |b| %>
<p>
<%= b.label :title%>
<%= b.text_field :title%>
</p>
<p>
<%= b.label :price%>
<%= b.text_field :price%>
</p>
<p>
<%= b.label :author%>
<%= b.text_field :author%>
```

```
</p>
<% end %>
```

Note:

1. Form automatically generated with the correct post URL for resource “book”
2. HTML Labels and Text Fields generated via corresponding tags. You can also use pure HTML
3. Tags available for other common HTML elements; provide convenient features like validation etc.

Layout Templates

Example Layout:

```
<!DOCTYPE html>
<html>
 <head>
      <!-- include scripts and stylesheets here -->
 </head>
<body>
      <div id=”header”>
            <!-- header goes here -- >
      </div>
      <div id=”main-body”>
            <%= yield %>
      </div>
      <div id=”footer”>
            <!-- header goes here -- >
      </div>
</body>
</html>
```

1. Each view can have a layout template, including no layout.
2. Typically used to create a template for header and footer of pages - then each view focuses on only its part
3. Layouts are in app/views/layouts; Default layout is “application.html.erb”
4. Include your views using `<%= yield %>` directive in layout
5. Controller wide layouts possible.
6. render directive also can take a layout parameter
    a. `render “new”, :layout => :home_page`
    b. `render “new”, :layout => false`

View Partials

Including repeated HTML code

Client Side Assets
Details of CSS, Javascript, and other client side assets will be covered in a later recitation.

Topic #3: Controllers
Controllers help isolate presentation layer from the data layer - separation of concerns
Using this - you can use the same code to develop a web based app, a mobile based app, or a some other view technology without changing much of the underlying code. Similarly you can use any data store - relational databases, flat files, etc. and switch between them without changing any of the view code.

Default/Base Controller: ApplicationController (application_controller.rb)

New controllers are created using:
```
$ rails generate controller <name> <action1> <action2> etc.
```

Lets look at books_controller.rb

```
class BooksController < ApplicationController

## This will use "app/views/layouts/book_layouts.html.erb" as the
## layout for all views in this controller, unless overriden in a
## method.
layout :book_layout

def index
 render "index", :layout => false
end

def new
 @time_now = Time.now
 ## render "new", :layout => false
end
end
```

You can start adding new methods which will perform actions. For example:

```
def checkout
      ## This is pseudo code
      @book = Book.find(params[:book_id])
      if (@book.available?)
          @book.checkout(params[:user_id])
```

```
            msg = "Book checked out"
      else
            msg = "Book not available for checkout"
      end
      render "checkout", :layout => false
end
```

Some useful objects available to you in the controllers are:
`params`: A hash of HTTP params which came with the request.
`request`: The HTTP request object as a hash
`session`: The session object as a hash

Some useful directives in controllers:

```
render "view", :layout => [layout_name | false]
render :nothing => true ## Useful for async actions for which you do
not need ack
render :inline => "<%= Time.now %>" ## I've never used this form
render :text => "OK" ## Useful for AJAX calls
render :json => @books
render :xml => @books
render :js => "<js_code_here>" ## Not a good practice

redirect_to "/relative/path/to/view"
```

Note: Any instance variables that you require in the view must be set up in the current action before calling render.

Topic #4: Data Access in Rails ("Model" part of MVC)

http://guides.rubyonrails.org/active_record_querying.html
http://guides.rubyonrails.org/migrations.html

ActiveRecord is a data interface/access layer: allows you to store/retrieve/query a database without having to learn and worry about SQL. All you need to know is the corresponding model class in ruby and rails takes care of the SQL level details of interaction with the database!

Generating a Model

```
$ rails generate model library name:string address:string
$  invoke  active_record
$      create    db/migrate/20130211132022_create_libraries.rb
$      create    app/models/library.rb
$      invoke    test_unit
```

```
$      create      test/unit/library_test.rb
$      create      test/fixtures/libraries.yml
```

This command generate the model class (app/models/library.rb), a migration script to create the corresponding table in the database (db/migrate/xxx_create_libraries.rb), and some files to run tests.

app/models/library.rb:

```
class Library < ActiveRecord::Base
 attr_accessible :address, :name
end
```

Note:

1. Library class extends from `ActiveRecord::Base` which provides many utility methods for interacting with Model objects. (as you will see)
2. `attr_accessible` directive specifies the accessible attributes of the model via getters and setters, i.e. “library.name”, “library.address” can be used on the LHS or RHS of an assignment expression.
3. Many more useful directives available.

db/migrate/20130211132022_create_libraries.rb:

```
class CreateLibraries < ActiveRecord::Migration
 def change
   create_table :libraries do |t|
     t.string :name
     t.string :address

     t.timestamps
   end
 end
end
```

Note:

1. This migration script creates the database table.
2. Automatically adds an “id” field to the model (not shown) and a timestamps field to hold the last updated timestamp for each record.

Creating the Database Table

```
$ rake db:migrate
```

Simple operations on the Library model

You can now start using this model in your code. Examples are shown below:

```
## Return a library object with id 10
@library = Library.find(10)

## Create a new library  object and save it to the database
lib = Library.new({:name => "Hayden", :address => "100 Memorial
Drive"})
lib.save()

## Retrieve a library object with name = "Hayden"
library = Library.find_by_name("Hayden")
```

More Fun Details of the Model

Enforcing Referential Constraints:
In Book class: `belongs_to :library`
In Library class: `has_many :books`

Retrieving multiple objects:
Use `Model.xxx where xxx = where, select, group, having, order, limit, join, order`

Topic #5: Some Useful Rails Configuration Files

1. `boot.rb, application.rb, environment.rb`
2. `database.yml`

Topic #6: Debugging Rails Apps

Which IDE should use for development?

Your TAs are familiar with the following:

- Sublime + Vi editor with command line
- Emacs Editor with command line
- RubyMine (30 day trial)

Others have used

- Aptana RadRails

For more of debugging, see: http://guides.rubyonrails.org/debugging_rails_applications.html

Topic #7: Some useful heroku commands

You'll only need to specify `--app <app_name>` if you're not in your app directory:

```
$ heroku logs [--app <app_name>]
```

For viewing application logs on heroku (use `$ heroku logs --tail` to stream the logs live)

```
$ heroku logs [--app <app_name>]
```

For viewing application logs on heroku

```
$ heroku config [--app <app_name>]
```

To configure your environment variables. Access to heroku services (mail server, data stores, social plugins etc) is usually via environment variables

```
$ heroku sql [--app <app_name>]
```

Provides a sql console to your database

```
$ heroku run rake db:create [--app <app_name>]
$ heroku run rake db:migrate [--app <app_name>]
$ heroku run ls -las [--app <app_name>]
$ heroku run ssh [--app <app_name>]
```

Runs the specified command. **Warning:** Use cautiously!

Next Tutorialp

Details of the Ruby programming language

6.170 Software Studio
Spring 2013