



Advanced Online Media

Dr. Cindy Royal

Texas State University - San Marcos

School of Journalism and Mass Communication

Intro to Ruby on Rails

Ruby on Rails is a Web Framework that relies on the Ruby language to develop a structure for advanced Web applications. When working in RoR, you typically set up a local development environment to test out and develop your applications, and then ultimately launch to a server on the Web. You can go to www.rubyonrails.org/download to learn more about what you need to install.

Another good installing option is railsinstaller.org. It sets up everything you need to get going in one simple install package.

Ruby

Ruby Gems

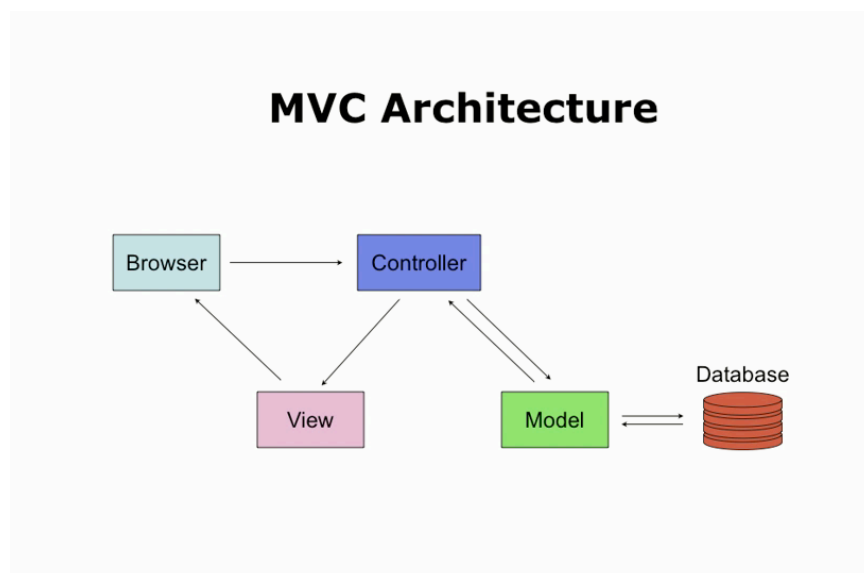
Rails

Database (sqlite3, mysql, postgres or other)

Terminal Application and Text Edit (or other text editor)

Most new Macs come with these items installed, so usually you are up and running in a few minutes. You may have to run through some upgrades for new versions. I have installed everything we need in the lab environment.

RoR relies on MVC architecture to develop applications:



MVC Architecture

Model – Responsible for maintaining state of application. It enforces the business rules that apply to data.

View – Responsible for generating a user interface, based on data in the model; doesn't handle data.

Controller – Orchestrates the application; receive events from the outside world, interact with the model, and display views.

When working in Rails, you will use a few different tools. You will be writing code to the command line via Terminal. This might seem a little odd to you at first, but you will get used to the Ruby code commands.

You will be modifying files in a text editor. You can use something like TextEdit or download an app with more features like TextMate or Komodo Edit.

We'll have a browser open, so we can test changes made to our application.

Start Your Application

To begin building an app, you simply login at the Terminal and switch to the folder where you want to host your files. The \$ indicates your login.

```
$ cd Sites
```

Then you create your rails app with the following:

```
$ rails new path/to/your/application
```

If you are already in the proper folder, then it looks like this:

```
$ rails new my_app
```

Rails creates your application. Then you can change to that folder.

```
$ cd my_app
```

At any point, if you want to look at the contents of a folder, use the ls command.

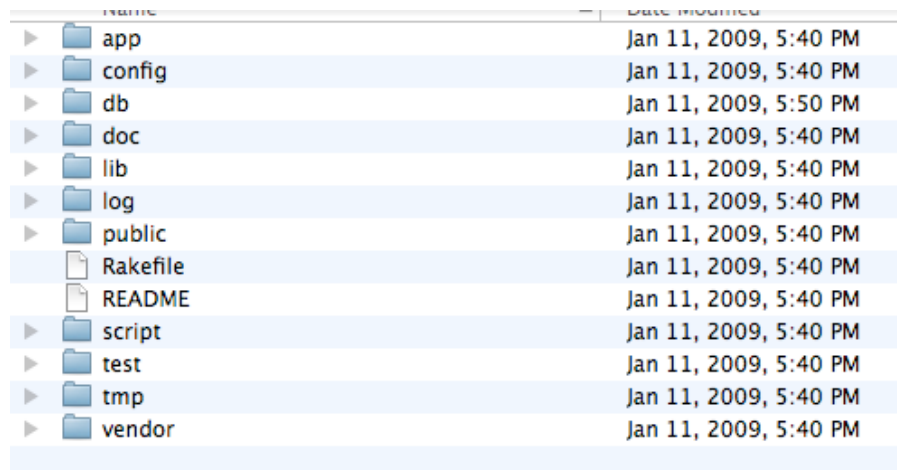
To start the server, run:

```
$ rails server
```

If you go to <http://localhost:3000> (or if you are on another server, go to that server's address and the 3000 port), you can see if your application has started. You should see the Welcome Aboard page. If so, you are on your way!

Leave the server running. If you ever need to stop the server, use Ctrl-C. You can open a New Terminal Shell to continue working. You will need to use the cd command to get into your apps folder.

Use the Finder to go to the my_app folder you just created on your computer. Take a look at the folder structure.



name	Date Modified
▶ app	Jan 11, 2009, 5:40 PM
▶ config	Jan 11, 2009, 5:40 PM
▶ db	Jan 11, 2009, 5:50 PM
▶ doc	Jan 11, 2009, 5:40 PM
▶ lib	Jan 11, 2009, 5:40 PM
▶ log	Jan 11, 2009, 5:40 PM
▶ public	Jan 11, 2009, 5:40 PM
Rakefile	Jan 11, 2009, 5:40 PM
README	Jan 11, 2009, 5:40 PM
▶ script	Jan 11, 2009, 5:40 PM
▶ test	Jan 11, 2009, 5:40 PM
▶ tmp	Jan 11, 2009, 5:40 PM
▶ vendor	Jan 11, 2009, 5:40 PM

Within the app folder, you will see
controllers
helpers
models
views

The controllers folder has a general application.rb file. Any controller you create will borrow from that file.

We will now generate a controller called Say.

```
$ rails generate controller Say hello goodbye
```

With your text editor, open the file say_controller.rb that RoR has just made

```
for you:
class SayController < ApplicationController

  def hello
  end

  def goodbye
  end

end
```

Find the files under app, views. You should find a folder called say. You will find the views for hello and goodbye methods. Go to this url and you should see the file for hello:

<http://localhost:3000/say/hello>

You have successfully created your first RoR file.

Note: .rb files are pure Ruby files. .html.erb files are a combination of some Ruby and some HTML. You may need to adjust your preferences in TextEdit for Open/Save to remove the Rich Text display for certain files.

The Browser sends request to Controller, Controller generates a View.

You can add the current time to the file by adding a paragraph to your View template like this:

```
<p>It is now <%= Time.now %><p>
```

The <% is where Ruby code is stored within an html document.

We could also put the time in the hello method in our controller.

```
class SayController < ApplicationController

  def hello
    @time = Time.now

  end
```

Where @time is a variable that you can now use in the View.

```
<p>It is now <%= @time %>
```

Linking pages

We can put links from the hello page to the goodbye page and vice versa this way:

```
<p> Say <a href="/say/hello">Hello</a>!
</p>
```

But a better, more flexible way is to use:

```
<p>Time to say <%= link_to "Goodbye", say_goodbye_path %>! </p>
```

The `link_to` includes the text for the link and calls the goodbye action. You can put a link to the Hello page in the Goodbye page.

Make a New Rails App and Connect to DB

Now, back to RoR. We are going to create a simple poll app. Make sure you are in the Sites folder (use `cd ..` to go back, `ls` to see what's in folder, `cd foldername` to change to folder)

```
$ rails new musicpoll
```

This creates the app, the default is using SQLite3 as database. If you have other databases installed, you can specify (ex. `-d mysql`)

Now change to the musicpoll directory

```
$ cd musicpoll
```

Then you can use this scaffolding command to generate a table with variables that connects to the app. Scaffold uses standard methods to interact with the database to make all the CRUD functionality.

CRUD: Create, Read, Update and Delete

```
$ rails generate scaffold Favorite name:string genre:string comments:string
```

Scaffold creates a "favorites" table with name, genre and comments as fields.

Then run the database migration:

```
$ rake db:migrate
```

Make sure your application is running

```
$ rails server
```

Go to <http://localhost:3000/favorites>
You will see the beginning of your application!

Adding data to the application

Got to the browser, and with the server running go to <http://localhost:3000/favorites>

Use the "New Favorite" link to add a new line to the database with user input. The data will be added to the database and you will see it in the list home page view. You can also see that scaffold has created the functionality for you to Show Edit Destroy the file, basically all the CRUD you need.

Validation

Let's make sure that people have to input certain things before they can go save a record. Go to the model in `apps/models/favorite.rb` and add this line

```
class Favorite < ActiveRecord::Base
  validates_presence_of :name, :genre
end
```

If anyone tries to create a record that doesn't have at least these 2 things, they will get an error.

Rails has lots of validation functions. For instance, we might want to make sure that duplicates aren't input (if we were doing some kind of an app that captured unique info)

```
validates_uniqueness_of :title
```

Advanced Rails Features and Modifications

Now that you have a working application, you need to add functionality, make it prettier. But the scaffolding feature gets you started quickly (rapid application development). There are many things you can do to the application now, some engaging your knowledge of html/css, some that will expand your programming knowledge.

Changes to Layout

The basic html template actually lives within the layouts folder under `app/views/layouts`. Open `application.html.erb`

You will see the reference to the stylesheet, which is under `app/assets/stylesheet`. Look for `scaffold.css` and modify.

Then, you can apply your own styling by manipulating the CSS and the HTML on various pages.

Let's make the table rows have alternating colors in the list.

If you look at index.html.erb (in app/views/favorites), you will see the table that is made when that page is run. It executes some ruby code that reads the data from the artists table into the table on the page based on how we set up the fields. There is one row in the code, because that repeats with the "do" loop. Add this class to <tr> so it will cycle alternating between two colors.

```
<tr class="<%= cycle('list-line-odd', 'list-line-even') %>">
```

Then, go into your scaffold.css stylesheet and add the following properties:

```
.list-line-even {  
  background: #e0f8f8;  
}
```

```
.list-line-odd {  
  background: #f8b0f8;  
}
```

Make dropdown for genre

We really want to create a dropdown for our genres, so we only get genres in which we are interested. Find _form.html.erb in the apps/views/favorites folder. Replace the genre text_field line with:

```
<%= f.collection_select :genre, Favorite::GENRES, :to_s, :to_s,  
:include_blank => true %>
```

Then in your model (apps/models/favorite.rb), include the line that defines your genres:

```
GENRES = ['Rock', 'Country', 'Classical', 'Pop', 'Other']
```

You can add this above the validation line.

Find total records

Count method counts records in db. Add this to the View index.html.erb (app/view/favorites/index.html.erb). Put this line below the table.

```
<p>Total records: <%= Favorite.count %></p>
```

Total the answers to genre

Now we want to total the results of genre so we can show in a chart.

First we adjust the model (app/model/favorite.rb)

Add these lines of code after the validation line:

```
scope :rockvotes, :conditions => { :genre => 'Rock' }  
scope :countryvotes, :conditions => { :genre => 'Country' }  
scope :classicalvotes, :conditions => { :genre => 'Classical' }  
scope :popvotes, :conditions => { :genre => 'Pop' }  
scope :othervotes, :conditions => { :genre => 'Other' }
```

They define the scope of each of our votes in genre based on the condition of the answer.

Then we adjust the controller (app/controllers/favorites_controller.rb):

In the def show area, add this (first line should be there):

```
@favorite = Favorite.find(params[:id])  
@rockvotes = Favorite.rockvotes.length  
@countryvotes = Favorite.countryvotes.length  
@classicalvotes = Favorite.classicalvotes.length  
@popvotes = Favorite.popvotes.length  
@othervotes = Favorite.othervotes.length
```

Then in the View for the Show page (app/view/favorites/show.html.erb)

```
<p>Total records: <%= Favorite.count %><br />  
Rock Votes: <%= @rockvotes %><br />  
Country Votes: <%= @countryvotes %><br />  
Classical Votes: <%= @classicalvotes %><br />  
Pop Votes: <%= @popvotes %><br />  
Other Votes: <%= @othervotes %></p>
```

Add these lines above the links at the bottom. I chose to enter these on the Show page, because ideally the user would never see and have access to the actual table with the CRUD on the index page.

Then you should be able to add a new vote and then see the results.

Show results in a chart

Last, but not least, let's show the results in a Google Chart. It's easy, now that all our variables are named.

```
<p>></p>
```

Put this code below the results and above the links in the show.html.erb page. When you paste, make sure there aren't any erroneous spaces created.

There's a lot more you can do to make a functioning and well designed app. But this is a start. Consider what you might be able to develop with just these basic skills.

Launching Your App on the Web

To launch your app on the Web, you need a server that supports Ruby and Rails and will support your database. Many hosts offer Ruby on Rails support, like your Bluehost account. You can also launch to a site like Heroku for simple testing or apps without heavy traffic anticipated. See Heroku Handout for more information.

We won't be doing these sections in class, but here are some additional instructions.

Add a New Genre

To add a new genre, you simply have to do all the things we did above to make sure your new genre is included.

Model (app/model/favorite.rb):

```
Hip Hop Votes: <%= @hiphopvotes %><br />
```

Controller (app/controllers/favorite_controller.rb):

```
@hiphopvotes = Favorite.hiphopvotes.length
```

View (app/view/favorites/show.html.erb):

```
<p>Hip Hop Votes: <%= @hiphopvotes %></p>
```

Add new genre to chart:

```
<p></p>
```

Modifying the App with Migrations

Migrations – add a column to the database table.

```
$ rails generate migration add_twitter_to_favorite twitter:string
```

The following migration is created in the db folder.

```
class AddTwitterToFavorite < ActiveRecord::Migration
  def change
    add_column :favorites, :twitter, :string
  end
end
```

Then run the migration again, if you made any changes to the file:

```
$ rake db:migrate
```

You need to make one change in the model. Add :twitter to the list of attr_accessible in favorite.rb.

For the view, you must change all the applicable files:

index.html.erb

Add a line for Twitter in the header of the table

```
<th>Twitter</th>
```

and a line to add to the table:

```
<td><%= favorite.twitter %></td>
```

form.html.erb - add at end of form, before submit div

```
<div class="field">
```

```
  <%= f.label :twitter %><br />
```

```
  <%= f.text_field :twitter %>
```

```
</div>
```

show.html.erb - slightly different; add to list

```
<p>
```

```
  <b>Twitter:</b>
```

```
  <%= @favorite.twitter %>
```

```
</p>
```