# 01 Introducing and installing Ruby gems and Ruby on Rails

### **Exercise** - Using ERB for templating

* Complete the exercise steps in this file and run the code  
  /Sites/Intro\_to\_Rails/L07\_ERB\_templating.rb

Additional detail on ERB templating can be found here:  
<http://www.stuartellis.eu/articles/erb/>

### **Exercise** - creating a Hello World application

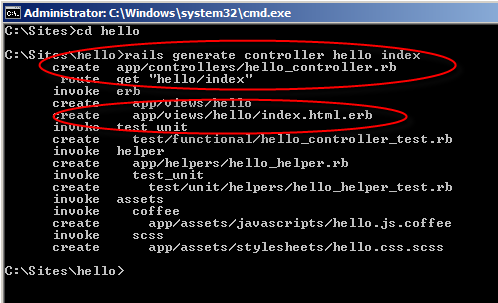
1. Open your terminal or command window
2. Navigate to your /Sites/ directory
3. Run this command  
   rails new hello
4. Review and consider the list of create and using statements which appear

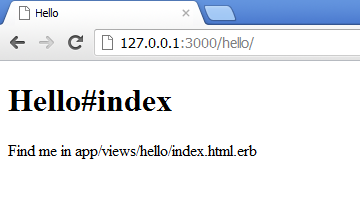
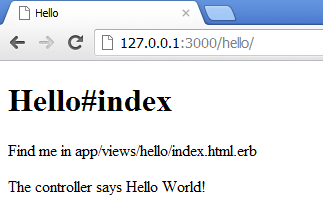
### **Exercise** - running and requesting a Hello World application

1. In a terminal or command window, navigate to your Rails sites directory  
   …/Sites
2. In the terminal or command window, change to the …/Sites/hello directory, and launch WEBrick  
   rails server
3. In a web browser, request the default page for your local HTTP server on port 3000  
   http://127.0.0.1:3000
4. You should the Rails Welcome Aboard page

### **Exercise** - creating an index page for the hello application

1. In a terminal or command window, navigate to your …/Sites/hello directory
2. Execute this command to generate a controller:  
   rails generate controller hello index
3. Review the statements describing the generated content



1. Open the application's routes.rb file and uncomment the default route
   1. File location: …/Sites/hello/config/routes.rb
   2. Route (at bottom of file): match ':controller(/:action(/:id))(.:format)'
2. Browse the URI for the view you just created:  
   <http://127.0.0.1:3000/hello>  
   
3. Open the controller you created, and declare an instance variable in the index method
   1. File location: …/Sites/hello/app/controllers/hello\_controller.rb
   2. Code example:  
        
      class HelloController < ApplicationController  
       def index  
       **@message = "Hello World!"**  
       end  
      end
4. Open the view you created, and add the following code to display your @message variable
   1. File location: …/Sites/hello/app/views/hello/index.html.erb
   2. Code example:  
        
      <p>The controller says <%= @message %></p>
5. In the browser, re-request the view you just created.  
   

# 02 Introducing controllers, layouts, views, and styling

### **Exercise** – configuring site wide and template specific css

1. Verify that
   1. You have successfully completed the prior exercise
   2. Your rails server is running for that application
2. Open and add the following CSS rule to application.css

body {

font-family:sans-serif;

}

1. Open and add the following CSS rule to hello.css.scss

h2 {

font-family:serif;

font-weight:bold;

color:#00FF00;

}

1. Open and add both <h2> and <p> text to app/views/index.html.erb
   1. You can generate entertaining dummy text here: <http://hipsteripsum.me/>
2. Browse the hello view to see the effect of your CSS
3. View source on the generated page. Drill through to the CSS and discuss what you find.

### Exercise – creating a view specific layout

1. Verify that
   1. You have successfully completed the prior exercise
   2. Your Rails server is running for that application
2. Edit app/views/layouts/application.html.erb to include this code  
   <h1>application.html.erb layout</h1>
3. Request the hello application  
   http://127.0.0.1:3000/hello/
4. Duplicate app/views/layouts/application.html.erb and rename it hello.html.erb
5. Modify hello.html.erb to include this code  
   <h1>hello.html.erb layout</h1>
6. Optional:
   1. in hello.html.erb, cut the stylesheet\_link\_tag
   2. save the change
   3. request the page to notice the change
   4. replace the stylesheet\_link\_tag
7. Discuss what you see with the instructor

### Exercise – Determining the layout from the controller

1. Verify that
   1. You have successfully completed the prior exercise
   2. Your Rails server is running for that application
2. Duplicate app/views/layouts/hello.html.erb and rename it admin.html.erb
3. Modify admin.html.erb to include this code  
   <h1>admin.html.erb layout</h1>
4. Duplicate app/views/layouts/admin.html.erb and rename it user.html.erb
5. Modify user.html.erb to include this code  
   <h1>user.html.erb layout</h1>
6. Edit app/controllers/hello\_controller.rb to include this code

class HelloController < ApplicationController

layout "admin"

def index

end

end

1. Request the hello application  
   http://127.0.0.1:3000/hello/
2. Notice which layout runs
3. Modify hello\_controller.rb to include this code  
   layout “user”
4. Request the hello application  
   http://127.0.0.1:3000/hello/
5. Notice which layout runs
6. Discuss what you see with the instructor

### Exercise – assigning layout content from its view

1. Open the template app/views/hello/index.html.erb
2. Add a content\_for block containing a title for this page

<% content\_for :page\_title do %>

Hello Page: Greetings!

<% end %>

1. Delete the page-specific layout for this template at app/views/layouts/hello.html.erb
2. Comment out the layout assignment in hello\_controller.rb  
   **#** layout “user”
3. Open the default layout for this template at app/views/layouts/application.html.erb
4. Modify the default template to yield the :page\_title content in its <title> block

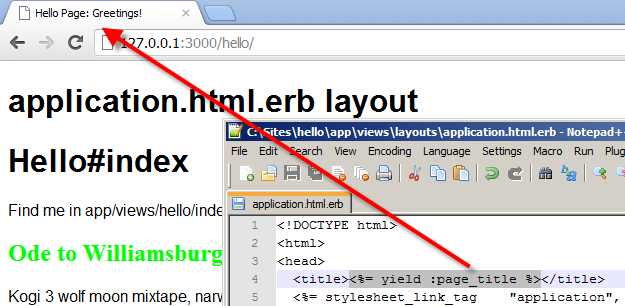
<!DOCTYPE html>

<html>

<head>

<title>**<%= yield :page\_title %>**</title>

1. Request the hello application  
   http://127.0.0.1:3000/hello/
2. Verify you see your :page\_title content displayed as the page title (e.g., on browser the tab)



### Exercise – changing the site root page to a controller and its view

1. Browse the root of the hello application  
   http://127.0.0.1:3000/
2. You should see the Rails “Welcome aboard” page
3. In the hello application, uncomment and modify the root in config/routes.rb to the hello#index action  
   root :to => ‘hello#index’
4. Restart the Rails server
5. Browse the root of the hello application  
   http://127.0.0.1:3000/
6. You should now see the hello#index page

# 03 Introducing Models and ActiveRecord

### Exercise – creating a simple input and display form

1. Create a new Rails application named bookreview  
   rails new bookreview
2. Change into the /Sites/bookreview directory
3. Generate a controller named books with an action named add  
   rails generate controller books add
4. Edit app/controllers/books\_controller.rb to add a book\_name instance variable, to be assigned from the view

def add

**@book\_name = params[:book\_name]**

end

1. Edit the view template for the add action at app/views/books/add.html.erb to submit and display a book\_name

<h2>Recent book: <%= @book\_name %></h2>

<%= form\_tag :action => 'add' do %>

<p>Book name: <%= text\_field\_tag 'book\_name', @book\_name %></p>

<%= submit\_tag 'Add Book' %>

<% end %>

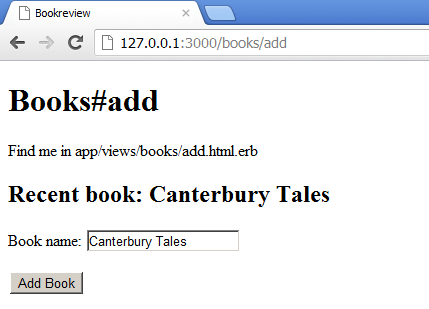
1. Edit config/routes.rb by uncommenting the default route match pattern

match ':controller(/:action(/:id))(.:format)'

1. Launch the rails server for the bookreview application
2. Request the add action of the books controller for the application

http://127.0.0.1:3000/books/add

1. Add a book. You should see the value you submit displayed after you press Add Book.



### Exercise – manually creating and using a model

1. Generate a book model for your application (remember: generate a model using a singular name)

rails generate model book

1. Edit the CreateBooks migration class at db/migrate/[timestamp]\_create\_books.rb to create a string column book\_name in the book table

class CreateBooks < ActiveRecord::Migration

def change

create\_table :books do |t|

**t.string :book\_name**

t.timestamps

end

end

end

1. Use the rake db:migrate command to cause Rails to create or rebuild the Books table

rake db:migrate

1. Edit the Book class derived from ActiveRecord::Base at app/models/book.rb to expose a book\_name accessor

class Book < ActiveRecord::Base

**attr\_accessible :book\_name**

end

1. Edit the BooksController at app/controllers/books\_controller.rb to create a new book unless book\_name is blank, and then retrieve all Book records and assign them to an instance variable named books

class BooksController < ApplicationController

def sign\_in

@book\_name = params[:book\_name]

**unless @book\_name.blank?**

**@book = Book.create({:book\_name => @book\_name})**

**end**

**@books = Book.all**

end

end

1. Edit the view template for the add action to display all saved book names, along with debug data

…

<p>Book list:</p>

<ul>

<% @books.each do |book| %>

<li><%= book.book\_name %></li>

<% end %>

</ul>

<hr>

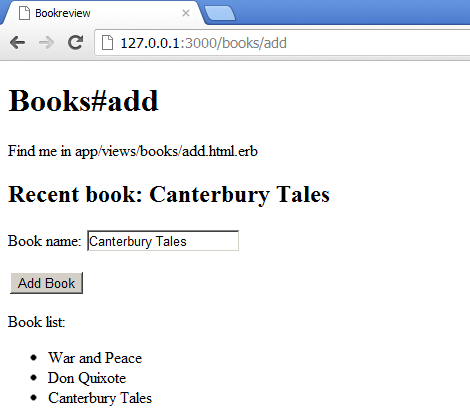
<%= debug(params) %>

<%= debug(assigns) %>

1. Request the add action of the books controller for the application

http://127.0.0.1:3000/books/add

1. Add a book. You should see all added books displayed.



# 04 Creating forms for data models

### Exercise – creating a Rails application using scaffolding

1. Create a new rails application named bookreview\_scaffold

rails new bookreview\_scaffold

1. Change into the new application directory
2. Generate scaffolding for a model named Book with one data field book\_name of the string data type

rails generate scaffold Book book\_name:string

1. Use Rake to generate tables based on the migration generated as part of the scaffolding

rake db:migrate

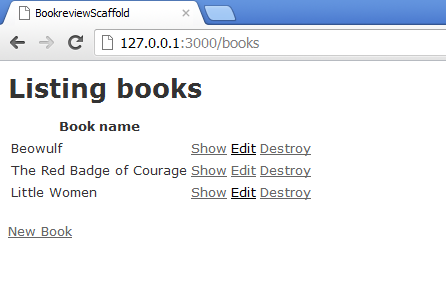
1. Launch the Rails server for the application

rails server

1. Browse the controller created by the scaffolding (remember: the generated controller is the plural of the model name)

http://127.0.0.1/books/

1. Add and edit some books



1. Review the generated code, specifically comparing these files between this application and the bookreview application created manually in the prior lesson

app/models/book.rb

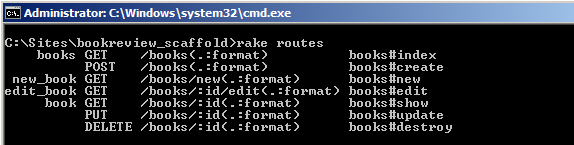
db/migrate/[timestamp]\_create\_books.rb

### Exercise – Understanding Rest-ful URIs

1. Verify that
   1. You have successfully completed the prior exercise
   2. Your Rails server is running for that application
2. Browse the Books controller

http://127.0.0.1/books

1. Open apps/controller/books\_controller.rb and examine each of the 7 actions (methods)
2. Create, edit, and destroy some books, while comparing the URI at each step to the route definitions show by rake routes



1. Modify the URI by adding a .json format extension, and notice the result

http://127.0.0.1/books.json

### Exercise – create and review multi-field forms

1. Create a new Rails application

rails new moviereview

1. In the moviereview directory, generate the scaffold for a Review model which can hold a movie title, synopsis, release date, release status, rating, and total movie production cost.

rails generate scaffold Review title:string rating:string synopsis:text release:date status:boolean score:integer cost:float

1. Use Rake to run the migration and create the database.

rake db:migrate

1. In the app/views/reviews directory, open the new.html.erb and edit.html.erb files and notice how they render \_form.html.erb
2. Open the \_form.html.erb file, and notice how it is wrapped in a form\_for command

### exercise – customizing a rails scaffolding form

1. Verify that
   1. You have successfully completed the prior exercise
   2. Your Rails server is running for that application
2. Launch the rails server for the moviereview application
3. Review app/views/reviews/\_form.html.erb and how it is included in other views
4. In the moviereview application, modify app/controllers/reviews\_controller.rb to initialize a hash of values named ratings

def initialize

@ratings = {

"G" => "General Admission",

"PG" => "Parental Guidance",

"R" => "Restricted",

"X" => "Adults Only"

}

end

1. Modify app/views/reviews/\_form.html.erb to comment out the default rating helper – a text\_field for string input – and replace it with a dynamically generated radio button set

<!--

<div class="field">

<%= f.label :rating %><br />

<%= f.text\_field :rating %>

</div>

-->

<fieldset>

<legend>Rating</legend>

<% @ratings.each do |label, value| %>

<%= f.radio\_button :rating, label %>

<label for="<%= 'movie\_rating\_' + value %>">

<%= label %>

</label>

<br/>

<% end %>

</fieldset>

1. Test your code to verify the radio buttons generate properly
2. Modify \_form.html.erb to test whether the :rating for a particular record being display in the form matches the code for the rating radio button being rendered for display

<% @ratings.each do |label, value| %>

**<% if :rating == value %>**

<%= f.radio\_button :rating, label, **:checked** %>

**<% else %>**

<%= f.radio\_button :rating, label %>

**<% end %>**

...

1. Test your code, create and save a review. Edit the review, and verify the saved rating is re-displayed in the form.
2. Comment out the radio button code, and replace it with a select helper method.

<%= f.select :rating, @ratings %>

1. Test your code.

### exercise – writing a customer helper method

1. Verify that
   1. You have successfully completed the prior exercise
   2. Your Rails server is running for that application
2. Declare a custom helper method for dynamically generating radio button sets in app/helpers/reviews\_helper.rb

module ReviewsHelper

def radio\_buttons(model\_name, source\_hash, target\_property, legend)

html = ''

html << '<fieldset><legend>' + legend + '</legend>'

source\_hash.each do |key, value|

html << radio\_button(model\_name, target\_property, value)

html << label(target\_property, value)

html << '<br/>'

end

html << '</fieldset>'

return html.html\_safe

end

end

1. In the \_form.html.erb partial, comment out the dynamically generated radio button set, and replace it with a call to the radio\_buttons helper method declared in the previous step.

<%= radio\_buttons("review", @ratings, :rating, "Rating") %>

1. Run the application. Verify the radio buttons generate, and that the correct radio button is re-selected when you edit a previously saved review.
2. Remove the .html\_safe assertion from the return statement in the radio\_buttons method.
3. Run the application. Notice that the radio button markup is escaped so that it displays instead of rendering the radio buttons. Discuss this behavior, and how it could prevent malicious script attacks.

# 05 Validating form input

### exercise – implementing validation helpers

1. Verify that
   1. You have successfully completed the previous exercise creating the moviereview application
   2. Your Rails server is running for that application
2. In app/models/review.rb, add validation requiring the presence of a value for the title field

validates\_presence\_of :title

1. Launch the application, and attempt to add a new review without a title value. You should see:



1. Open the form partial at app/views/reviews/\_form.html.erb, and review the code which displays this error message.
2. In app/models/review.rb, add validation to require a rating, with a custom message

validates\_presence\_of :rating, :message => “is needed, please add one!”

1. Add validation to ensure score is a value between 1 and 10

validates\_numericality\_of :score, :greater\_than => 0, :less\_than\_or\_equal\_to => 10

1. Test the application by attempting various input combinations which violate your validation rules.
2. Modify the score validation so that it is only enforced for new reviews.
3. Test the application and verify you can set a score higher than 10, but only when editing an existing review.

### exercise – conditionally validating fields

1. Verify that
   1. You have successfully completed the previous exercise
   2. Your Rails server is running for that application
2. Modify the review model to require a synopsis for G and PG movies, permit one for R, and forbid one for X.
3. Test the application and verify synopses are required, permitted, and forbidden as appropriate.

# 06 Handling file uploads and automating form building

### Exercise – implementing file upload and display

1. Verify that
   1. You have successfully completed the previous exercise creating the moviereview application
   2. Your Rails server is running for that application
2. Open the partial at app/views/reviews/\_form.html.erb, and add a label and file\_field control for :image

<div class="field">

<%= f.label :image %><br/>

<%= f.file\_field :image %>

</div>

1. Modify the form\_for helper to generate HTML encoding for multipart/form-data

<%= form\_for(@review, :html => { :multipart => true }) do |f| %>

1. In the terminal or command window, generate a migration add\_image\_type\_to\_review

rails generate migration add\_image\_type\_to\_review

1. Open db/migrate/[timestamp]\_add\_image\_type\_to\_review.rb, and modify its change method to add a :string data type :image\_type column to the :reviews table

def change

add\_column :reviews, :image\_type, :string

end

1. In the terminal or command window, execute the new migration

rake db:migrate

1. Open app/models/review.rb, and assign :store\_image to the after\_save callback, so that a method of this name (to be written below) will be called after each save of the model’s data to the database

after\_save :store\_image

1. Define a constant IMAGE\_STORE to hold the path to a public folder to be named image\_store relative to Rails.root

IMAGE\_STORE = "#{Rails.root}/public/image\_store"

1. Add a method named image\_filename which generates and returns a path and filename based on the ID of the review, and the image type of the uploaded image (image\_type is determined and stored in another method below)

def image\_filename

return "#{IMAGE\_STORE}/#{id}.#{image\_type}"

end

1. To prevent external code from calling this method, add the private keyword, and below it declare a store\_image method to write uploaded file\_data to the IMAGE\_STORE path

# after saving other date, store image on file system

# mark store\_image method private  
private

def store\_image

if @file\_data

# create directory at IMAGE\_STORE if it does not exist

FileUtils.mkdir\_p IMAGE\_STORE

# save image to file location and name from image\_filename method

File.open(image\_filename, 'wb') do |f|

f.write(@file\_data.read)

end

# nil file\_data in memory so it won't be resaved

@file\_data = nil

end

end

1. Above the private keyword – so that external code can access it – declare an image=(file\_data) method, which will be called when data is assigned to the image property of the model

# at upload assign file\_data to instance var and grab image type

def image=(file\_data)

unless file\_data.blank?

# assign uploaded data to instance variable

@file\_data = file\_data

# assign image type (extension) to self.image\_type

self.image\_type = file\_data.original\_filename.split('.').last.downcase

end

end

1. Above the private keyword, declare a image\_uri method to return a URI for this model’s image

# define method to retrieve relative URI for the stored image

# in the /public directory of the app, for use in HTML

def image\_uri

return "/image\_store/#{id}.#{image\_type}"

end

1. Above the private keyword, declare a has\_image? Method to return true|false whether this model has an image

# define method to determine if a review has an image on the

# file system at the location specified by image\_filename

def has\_image?

return File.exists? image\_filename

end

1. Open app/views/reviews/show.html.erb, and modify its code to display an uploaded image for a review, if any, or the string “No image available” if there is none

<p>

<b>Image:</b>

<% if @review.has\_image? %>

<%= image\_tag @review.image\_uri %>

<% else %>

No image available

<% end %>

</p>

1. Verify you have an image on your file system to use for testing. Run the application. Write a new review with an uploaded image, and verify you can edit and show the review with the image.

### exercise – customizing formbuilder to create a new form element

1. Verify that
   1. You have successfully completed the previous exercise creating the moviereview application
   2. Your Rails server is running for that application
2. Generate a new migration to add a :country field to the reviews model

rails generate migration add\_country\_to\_review

1. Modify db/migrate/[timestamp]\_add\_country\_to\_review.rb to add a :string column :country to the :reviews table

class AddCountryToReview < ActiveRecord::Migration

def change

add\_column :reviews, :country, :string

end

end

1. Use rake to run this database migration

rake db:migrate

1. In app/helpers, create a new file custom\_form\_helper.rb, and within it derive the FormBuilder class as a new class called CustomFormBuilder

class CustomFormBuilder < ActionView::Helpers::FormBuilder

end

1. In CustomFormBuilder, declare a create\_countries\_array method which populates an instance variable named @countries with a two-dimensional array of country names (the data structure used by a FormBuilder select helper)

def create\_countries\_array

@countries = Array.new

@countries.push(["Australia","Australia"])

@countries.push(["Canada","Canada"])

@countries.push(["Russia", "Russia"])

@countries.push(["United Kingdom","United Kingdom"])

@countries.push(["United States of America","United States of America"])

end

1. Declare a method named country\_select which calls the method to create the @countries array, then generates a select control, pre-populated by that array

def country\_select(method, options={}, html\_options={})

create\_countries\_array

select(method, @countries, options, html\_options)

end

1. Modify the form\_for element of the partial \_form.html.erb to use the custom form builder

<%= form\_for(@review, :html => { :multipart => true }),

:builder => CustomFormBuilder do |f| %>

1. Modify the partial \_form.html.erb to display the :country column using the country\_select element

<div class="field">

<%= f.label :country %><br/>

<%= f.country\_select :country %>

</div>

1. Modify the show.html.erb view to display country data for the review model

<p>

<b>Country:</b>

<%= @review.country %>

</p>

1. Test the application to verify you can select and display a country for each review.

### exercise – extending an existing form helper method to add features

1. Verify that
   1. You have successfully completed the previous exercise creating the moviereview application
   2. Your Rails server is running for that application
2. In app/helpers/custom\_form\_builder.rb, extend the label helper to conditionally add a large red asterisk next to the label, if the label helper is flagged :required => true

def label(method, options={}, html\_options={})

html\_to\_add = ""

if options[:required]

html\_to\_add = "<span style='color:red;font-size:18'>\*</span>"

end

super(method, options) + html\_to\_add.html\_safe

end

1. In the partial \_form.html.erb, modify the label for the :title field to mark it as required

<div class="field">

<%= f.label :title, :required => true %><br />

<%= f.text\_field :title %>

</div>

1. Run the application to verify the :title field is marked with a red asterisk

# 07 Creating multi-model associations

### exercise – creating and using two models in one form

1. Create a new Rails application named moviereview2

rails new moviereview2

1. Generate scaffolding for a Movie model

rails generate scaffold Movie title:string synopsis:text release:date

1. Generate scaffolding for a Review model, ensuring it has a movie\_id:integer column, to hold references to a particular movie

rails generate scaffold Review author:string review:text score:integer movie\_id:integer

1. Open app/models/movie.rb and associate this model with the :reviews

class Movie < ActiveRecord::Base

has\_many :reviews

end

1. Open the app/models/review.rb, and associate this model with a :movie

class Review < ActiveRecord::Base

belongs\_to :movie

end

1. Generate the database tables corresponding to these models

rake db:migrate

1. Start the Rails server

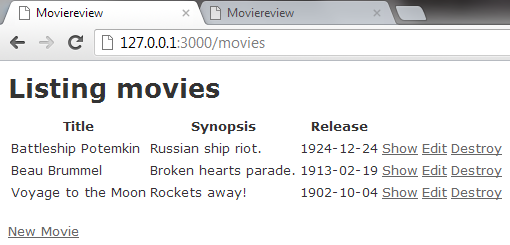
rails server

1. Modify app/views/movies/\_form.html.erb to support a starting year of 1900

<%= f.date\_select :release, :start\_year => 1900 %>

1. Using a browser, launch the Movies interface in your application, and create three movie records

http://127.0.0.1:3000/movies



1. Modify app/views/reviews/\_form.html.erb to replace the existing movie\_id field with a select control which displays all the movies in the Movies model

<%= f.select :movie\_id, Movie.find(:all, :order => "title").collect { |movie| [movie.title, movie.id]} %>

1. Load the Reviews interface, add a new review, and verify you can select a movie by its title

http://127.0.0.1:3000/reviews/new

1. Modify app/models/movie.rb to return a title\_date value display the combined movie title and release date

class Movie < ActiveRecord::Base

has\_many :reviews

def title\_date

return self.title + " (" + self.release.to\_s + ")"

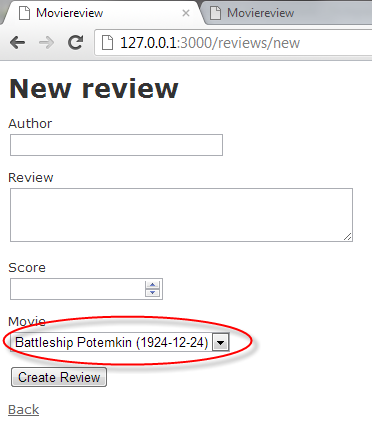
end

end

1. Modify app/views/reviews/\_form.html.erb to display the combined title and release date, instead of just the title

<%= f.select :movie\_id, Movie.find(:all, :order => "title").collect { |movie| [movie.title\_date, movie.id]} %>

1. Test the application to verify you can enter a review for your movie, selected by its title and release date



### exercise – validating the existence of related records

1. Verify that
   1. You have successfully completed the previous exercise creating the moviereview2 application
   2. Your Rails server is running for that application
2. Load the Movies interface, and add another movie

http://127.0.0.1:3000/movies

1. Run the Rails DB Console

rails dbconsole

1. View the contents of the movies table, and note the ID of the movie just added

select \* from movies;

1. Load the Reviews interface, write a review for the new movie, but do not save it yet
2. In the Movies interface, destroy the movie for which your review is pending
3. In the Reviews interface, save the review, and notice there is no error message
4. In the Rails DB Console, views the contents of the movies and reviews table, and notice there is now a review associated with a movie record which no longer exists

select \* from reviews;

select \* from movies;

1. Using the terminal or command window, install the validates\_existence gem

gem install validates\_existence

1. Modify moviereview2/gemfile to add an entry for this new gem

gem 'validates\_existence', '>= 0.4'

1. In app/models/review.rb, add a command to validate the existence of a movie before adding a record related to it (along with a setting to suppress duplicate error messages)

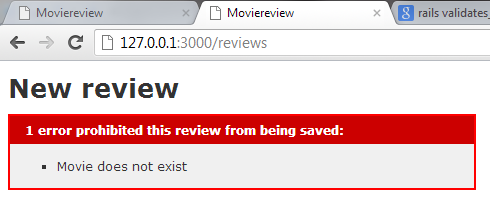
class Review < ActiveRecord::Base

belongs\_to :movie

validates\_existence\_of :movie, :both => false

end

1. In the Movies interface, add another movie
2. In the Reviews interface, write a review for the new movie, but do not add it yet
3. In the Movies interface, destroy the newly added movie
4. In the Reviews interface, try to add the review, and notice the error message you now receive



1. Use Rails DB Console to delete the orphaned review (delete by its ID, the first value seen when selecting table values for display)

delete from reviews where ID = 1

### exercise – listing and destroying related records

1. Verify that
   1. You have successfully completed the previous exercise creating the moviereview2 application
   2. Your Rails server is running for that application
2. Modify app/models/movie.rb to destroy related reviews when a movie record is destroyed

has\_many :reviews, :dependent => :destroy

1. Test the application by adding a new movie and a related review. Destroy the movie, then verify the related review was also destroyed.
2. Modify app/views/movies/index.html.erb to list the count of reviews for each movie

<th>Reviews</th>

...

<td><%= movie.reviews.count %></td>

1. Modify app/views/movies/index.html.erb to provide a link to the reviews list using its route

<%= link\_to 'Reviews', reviews\_path %>

1. Modify app/views/reviews/index.html.erb to provide a link to the movies list using its route

<%= link\_to 'Movies', movies\_path %>

1. Modify app/views/movies/show.html.erb to list a movie’s reviews when viewing the movie record

<h3>Reviews</h3>

<table>

<tr>

<th>Author</th>

<th>Score</th>

</tr>

<% for review in @movie.reviews %>

<tr>

<td><%= review.author %></td>

<td><%= review.score %></td>

</tr>

<% end %>

</table>

1. Test the application by adding reviews to a movie, and verifying the movie’s review count and review list are displayed

### exercise – drilling down from a summary to a detail view

1. Verify that
   1. You have successfully completed the previous exercise creating the moviereview2 application
   2. Your Rails server is running for that application
2. Modify config/routes.rb to add a route supporting a URI for the reviews associated with each movie

# resources :movies

resources :movies do

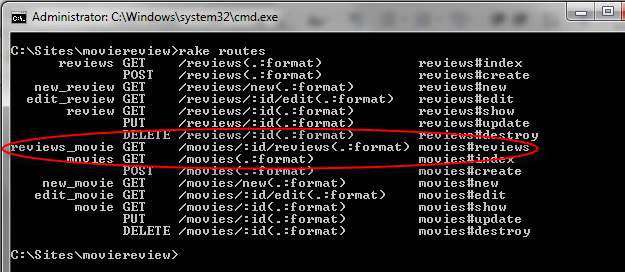
member do

get 'reviews'

end

end

1. In the terminal or command window, run the rake routes command to observe the new route



1. Modify app/controllers/movie\_controller.rb to support a reviews action which uses the movie identified by the :id parameter, gathers its reviews, then renders the reviews index view instead of the movies index view

def reviews

@movie = Movie.find(params[:id])

@reviews = @movie.reviews

respond\_to do |format|

format.html { render 'reviews/index' } # index.html.erb

format.json { render json: @movies }

end

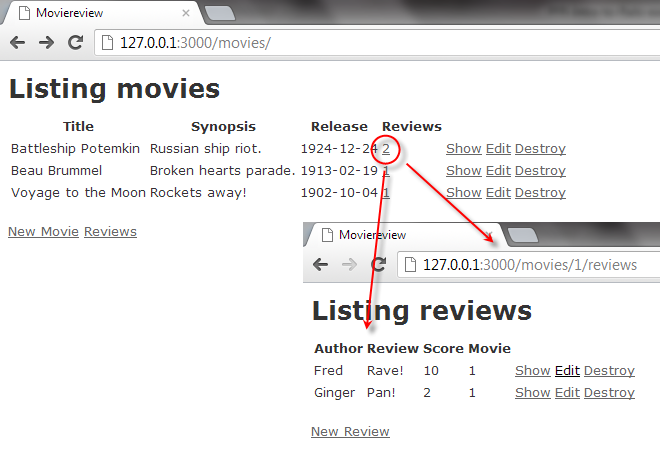
end

1. Modify app/views/movies/index.html.erb replacing the display of the reviews count with a link displaying the reviews count and also linking to the new route

<!-- <td><%= movie.reviews.count %></td> -->

<td><%= link\_to movie.reviews.count, reviews\_movie\_path(movie.id) %></td>

1. Launch the movies listing and verify that the reviews count is linked to a filtered list of reviews associated with each movie



### Exercise – adding site wide navigation

1. Verify that
   1. You have successfully completed the previous exercise creating the moviereview2 application
   2. Your Rails server is running for that application
2. Create a new directory and view partial named app/views/application/\_navigation.html.erb
3. Cut and paste the movies and reviews navigation links from their respective view index pages into this new file, wrapped in a paragraph tag

<p>

<%= link\_to 'Movies', movies\_path %>

<%= link\_to 'Reviews', reviews\_path %>

</p>

1. Modify app/views/layouts/application.html.erb to render the navigation partial in the site wide application layout, between the body and yield tags

...

<body>

**<%= render 'navigation' %>**

<%= yield %>

1. Run the application and test the navigation

### Exercise – enabling many to many associations

1. Verify that
   1. You have successfully completed the previous exercise creating the moviereview2 application
   2. Your Rails server is running for that application
2. Create scaffolding (and underlying model) for festival with a single string column name

rails generate scaffold festival name:string

1. Create a migration to join the movie and festival tables

rails generate migration CreateFestivalsMovies

1. Modify db/migrate/[timestamp]\_create\_festivals\_movies.rb to create, index, and drop the festivals\_movies table

class CreateFestivalsMovies < ActiveRecord::Migration

def up

create\_table :festivals\_movies, :id => false do |t|

t.integer :festival\_id, :null => false

t.integer :movie\_id, :null => false

end

# add index to optimize speed

add\_index :festivals\_movies, [:festival\_id, :movie\_id], :unique => true

end

def down

remove\_index :festivals\_movies, :column => [:festival\_id, :movie\_id]

drop\_table :festivals\_movies

end

end

1. Run rake db:migrate to generate the festivals and festivals\_movies tables

rake db:migrate

1. Modify app/models/movie.rb to establish its association with the festival model

has\_and\_belongs\_to\_many :festivals

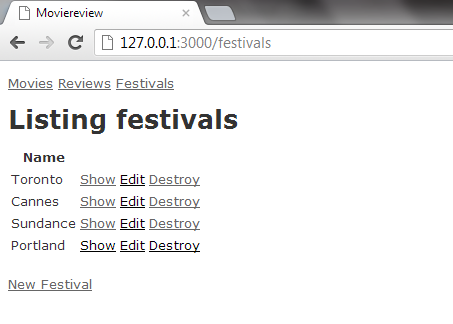
1. Modify app/models/festival.rb to establish its association with the movie model

has\_and\_belongs\_to\_many :movies

1. Modify app/views/application/\_navigation.html.erb to add a link to the Festivals interface

<%= link\_to 'Festivals', festivals\_path %>

1. Launch the Festivals interface and add three festival names



# 08 Debugging and testing in Rails

### Exercise – Writing values to the log file

1. Verify that
   1. You have successfully completed the prior exercise creating the moviereview2 application
   2. Your Rails server is running for that application
2. Modify the edit action in app/controllers/movie\_controller.rb to log the select movie title

# GET /movies/1/edit

def edit

@movie = Movie.find(params[:id])

logger.info "\*\*\* " + @movie.title + " selected for editing \*\*\*"

end

1. In the Movies index view, select a movie to edit
2. Open log/development.log, review the most recent output and find the log message

### Exercise – displaying variables as debug output

1. Verify that
   1. You have successfully completed the prior exercise creating the moviereview2 application
   2. Your Rails server is running for that application
2. Modify the reviews action in movies\_controller to raise and display the selected reviews

def reviews

@movie = Movie.find(params[:id])

@reviews = @movie.reviews

raise @reviews.to\_yaml

respond\_to do |format|

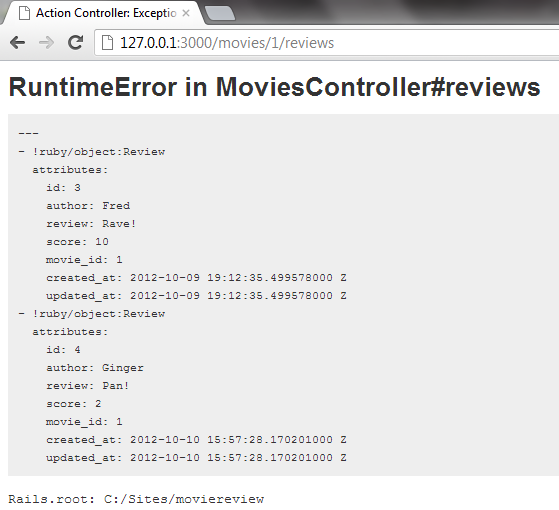
format.html { render 'reviews/index' } # index.html.erb

format.json { render json: @movies }

end

end

1. In the Movies index view, select the filtered reviews for a movie, and you should see an error with the reviews variable values



1. Modify app/views/movies/index.html.erb to display all movies as debug output

<br />

<%= link\_to 'New Movie', new\_movie\_path %>

<%= debug(@movies) %>

1. Select the Movies index and view the debug output

### Exercise – Install, configure, and use the Ruby debugger

1. Open a terminal or command window and install the Ruby debugger

gem install ruby-debug19

1. Configure the Gemfile to enable debugging

gem ‘ruby-debug19’, :require => ‘ruby-debug’

1. In app/controllers/festivals\_controller.rb, set a debugger breakpoint

def index

@festivals = Festival.all

debugger

respond\_to do |format|

format.html # index.html.erb

format.json { render json: @festivals }

end

end

1. Launch the Rails server in debug mode

rails server --debugger

1. Examine the server output during debugging
2. Use the list, next, and cont commands

### Exercise – building and running a simple unit test

1. Verify that
   1. You have successfully completed the prior exercise creating the moviereview2 application
   2. Your Rails server is running for that application
2. Open the application Gemfile and uncomment the following line to enable the BCrypt library

# To use ActiveModel has\_secure\_password

gem 'bcrypt-ruby', '~> 3.0.0'

1. Modify app/models/review.rb to remove validates\_existence :movie, and add validation to require a score between 0 and 10

class Review < ActiveRecord::Base

belongs\_to :movie

# validates\_existence\_of :movie, :both => false

validates\_presence\_of :score

validates\_numericality\_of :score, :greater\_than => 0, :less\_than\_or\_equal\_to => 10

end

1. Modify test/unit/review\_test.rb to test if a review with a too-high score is not valid

require 'test\_helper'

class ReviewTest < ActiveSupport::TestCase

def test\_score\_too\_high

# test if a too high score is not valid

review = Review.new

review.score = 15

assert !review.valid?, "Failed because too high score was found valid"

end

end

1. Modify test/unit/review\_test.rb to test if a review with an in-range score is valid

def test\_score\_just\_right

# test if an in-range score is valid

review = Review.new

review.score = 8

assert review.valid?, "Failed because in-range score found not valid"

end

1. Using the terminal or command window, run the tests and verify both succeed

rake test:units

1. Modify app/models/review.rb to remove validation for the score value

class Review < ActiveRecord::Base

belongs\_to :movie

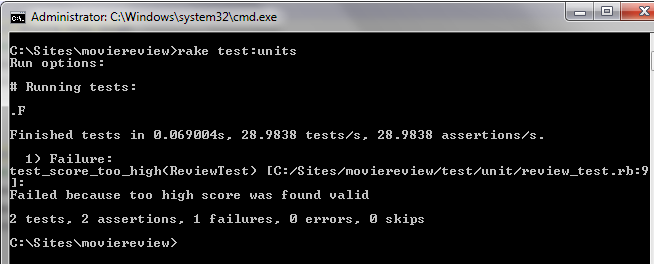
# validates\_existence\_of :movie, :both => false

validates\_presence\_of :score

# validates\_numericality\_of :score,:greater\_than => 0,:less\_than\_or\_equal\_to => 10

end

1. Re-run the tests, and verify the too-high test fails



# 09 Authenticating and managing users

### Exercise – setting and accessing cookies

1. Verify that
   1. You have successfully completed the prior exercise creating the moviereview2 application
   2. Your Rails server is running for that application
2. Generate a new controller and view named login

rails generate controller login login

1. Enable default routing in config/routes.rb by uncommenting this line

match ':controller(/:action(/:id))(.:format)'

1. Modify app/views/login/login.html.erb to display this form

<h2>Hello <%= @memory\_name %></h2>

<%= form\_tag :action => 'login' do %>

<p>What is your name?</p>

<%= text\_field\_tag 'input\_name', @memory\_name %>

<%= submit\_tag 'Log In' %>

<% end %>

<% unless @previous\_memory\_name.blank? %>

<p>Last time your name was: <%= @previous\_memory\_name %></p>

<% end %>

1. Modify app/controllers/login\_controller.rb to support this login action

def login

@previous\_memory\_name = cookies[:stored\_name]

@memory\_name = params[:input\_name]

cookies[:stored\_name] = @memory\_name

end

1. Launch and interact with the login form

http://127.0.0.1:3000/login/login

1. Use your browser to inspect cookies set by this page
   1. Chrome: (right-click) > Inspect element > Resources > Cookies > 127.0.0.1
   2. Firefox: (right-click) > View Page Info > Details > View Cookies

### Exercise – using session data

1. Verify that
   1. You have successfully completed the prior exercise creating the moviereview2 application
   2. Your Rails server is running for that application
2. Generate a new controller named list with a view named index

rails generate controller list index

1. Modify app/controllers/list\_controller.rb to create and manage a session array named list

class ListController < ApplicationController

def index

# retrieve items from session

@items = session[:items]

# if no items arrray, create it

unless @items

@items = Array.new

end

# add the item sent by the form, if any, to the items array

if params[:item]

@items << params[:item]

end

# update the session items

session[:items] = @items

end

end

1. Modify app/views/list/index.html.erb to accept and display list items

<h2>To Do List</h2>

<%= form\_tag :action => 'index' do %>

<p>What goes on the list?</p>

<%= text\_field\_tag 'item' %>

<%= submit\_tag 'Add item' %>

<% end %>

<% if @items %>

<ol>

<% @items.each do |item| %>

<li><%= item %></li>

<% end %>

</ol>

<% end %>

1. Launch and interact with the list application

http://127.0.0.1/list

1. Keep the browser open, but close the page, then re-open the page
2. Shut down the browser, then re-launch the application

### Exercise – implementing rails authentication

1. Verify that
   1. You have successfully completed the prior exercise creating the moviereview2 application
   2. Your Rails server is running for that application
2. Generate a user model with email and password\_digest properties

rails generate model user email:string password\_digest:string

1. Use rake to run the migration

rake db:migrate

1. Modify app/models/user.rb to expose email and password attributes

attr\_accessible :email, :password

1. Configure the user model to have a secure password

has\_secure\_password

1. Add presence and uniqueness validation to the user model

validates\_presence\_of :email

validates\_presence\_of :password, :on => :create

validates\_uniqueness\_of :email, :case\_sensitive => false

1. Generate a users controller and view with new and create actions

rails generate controller users new create

1. Modify app/controllers/users\_controller.rb new action to create a new user

def new

@user = User.new

end

1. Modify app/controllers/users\_controller.rb create action to create a User and hold its ID in the session

def create

@user = User.new(params[:user])

if @user.save

session[:user\_id] = @user.id

redirect\_to movies\_path

else

render 'new'

end

end

1. Generate a sessions controller and view with new, create, and destroy actions

rails generate controller sessions new create destroy

1. Modify config/routes.rb to map sessions/new as route named 'log\_in'

get "sessions/new", :as => :log\_in

1. Change sessions/create to a post and sessions/destroy to a delete

post "sessions/create"

delete "sessions/destroy"

1. Replace the get users/new and users/create routes with resources routes

# get "users/new"

# get "users/create"

resources :users

1. Modify the app/controllers/sessions\_controller.rb destroy action to reset the session

and redirect the user to the log\_in path

def destroy

reset\_session

redirect\_to log\_in\_path

end

1. Modify the create action to find the user, authenticate, set their ID in the session, then redirect to the movies path if possible, else render the new user view if they do not authenticate

def create

if user = User.find\_by\_email(params[:email]).try(:authenticate, params[:password])

session[:user\_id] = user.id

redirect\_to movies\_path

else

render 'new'

end

end

1. In app/views/sessions/new.html.erb add a form to accept an email and password to create a new session for the user, if they authenticate

<h2>Login</h2>

<%= form\_tag '/sessions/create' do %>

<%= text\_field\_tag :email %>

<%= password\_field\_tag :password %>

<%= submit\_tag 'Login' %>

<% end %>

1. Run the rails console and create a User with email 'test@test.com' and password 'password'

rails console

...

irb(main): User.create email:'test@test.com', password:'password'

1. Verify the rails server is running for the application
2. Launch the sessions/new action in a browser to view the Login form

http://127.0.0.1:3000/sessions/new

1. In the Login page, enter invalid credentials: email 'test@test.com' password 'wrong'. The Login page should be redisplayed
2. Enter the valid User credentials added through the console: email 'test@test.com' password 'password'. You should be redirected to the Movies views
3. Modify app/controllers/application\_controller.rb to add a current\_user method to return the current user, if there is one

def current\_user

if session[:user\_id]

# ||= assigns only if not already assigned (only calling db if necessary)

@current\_user ||= User.find(session[:user\_id])

end

@current\_user

end

1. Add a method logged\_in which returns a boolean true|false of whether a current user exists and therefore the user has logged in

def logged\_in?

# !! Ruby trick - double-negate current\_user existence and return the boolean

!!current\_user

end

1. Add a method require\_user which performs a redirect to the log\_in path unless the user is logged in

def require\_user

redirect\_to log\_in\_path unless logged\_in?

end

1. Make these methods available to all controllers and views by assigning them as helper methods within the application controller

helper\_method :current\_user, :logged\_in?, :require\_user

1. Secure the movies, reviews, and festivals controllers by adding the require\_user method as a before\_filter for each of these three controllers

class MoviesController < ApplicationController

before\_filter :require\_user

...

class ReviewsController < ApplicationController

before\_filter :require\_user

...

class FestivalsController < ApplicationController

before\_filter :require\_user

...

1. Close and re-open the browser to clear the current session
2. Attempt to browse the movies controller, and you should be redirected to the Login page

http://127.0.0.1:3000/movies

1. Verify the reviews and festivals controllers are also secured
2. Login and you should be redirected to the movies controller
3. Modify app/views/users/new.html.erb to create a Register form to populate a new User

<h2>Register User</h2>

<%= form\_for @user do |f| %>

<%= f.text\_field :email %>

<%= f.password\_field :password %>

<%= submit\_tag 'Register' %>

<% end %>

1. Modify the app/views/application/\_navigation.html.erb partial to add a Log Out link which calls the destroy action of the sessions controller, but appears only if user is logged in

<%= link\_to 'Log Out', '/sessions/destroy', {:method=>:delete} if logged\_in? %>

1. Add a Register link to the new user path which appears unless the user is already logged in

<%= link\_to 'Register', new\_user\_path unless logged\_in? %>

1. After the links, display the current user's email if they are signed in

<%= current\_user.email if logged\_in? %>

1. Attempt to browse the movies controller, and you should be redirected to the Login page, and you should see the Register link

http://127.0.0.1:3000/movies

1. Follow the Register link, and register a new user: email 'test2@test.com' and password 'password'. Once registered, the user should be logged in
2. Verify you can access all controllers using the new log in and that you see the current user email address displayed with a Log Out link
3. Log out of the application, and log in again with the previously registered email and password