CHAPTER 4

# Rails Views

## What are actionviews?

Views are the visual presentation element of the Rails application.

They can be found in the \app\views directory of your application.

Their naming convention is usually:

**index.html.erb**

The .erb shows that these are 'EmbeddedRuBy' files.

They are effectively template files for generating the HTML that the application needs to return to the client browser.

They usually contain a mixture of 'pure' HTML and tagged Ruby code.

When the controller 'calls' the view the embedded Ruby code is auctioned and replaced with the relevant HTML and then the controller 'sends' the page via the HTTP server back to the client.

As with the controller files this means that the erb files can be from very simple, maybe containing no embedded Ruby, to very complex with almost no 'pure' HTML and everything generated by the embedded Ruby code.

The embedded Ruby code is identified by the <% %> combination, if you are familiar with PHP for example these tags act like the <?php ?> tags.

The Ruby tags can appear anywhere within the file and even within valid HTML tags.

A simple example of a basic index.html.erb file is shown below.

This was generated automatically by running **rails g controller Employee index** from within the rails application directory.

<h1>Employee#index</h1>

<p>Find me in app/views/employee/index.html.erb</p>

As you can see it is just standard HTML coding.

The 'link' to this page for the application is via the controller method entry.

**def index**

**end**

There is no code in the method so Rails can only render the basic page.

If we add some Ruby code to the view and the controller, we can make the view a little more interesting.

The following shows and example of this:

### Controller code – (previously generated by rails)

By adding the line to the def index method we are linking the controller to the incoming request and asking it to get the data value associated with a field called visitor\_name.

**def index**

**@name = params[:name]**

**end**

### The view code – (previously generated by rails)

Replace the **<h1>Employee#index</h1**> block with a friendlier heading.

**<h1> Hello <%= @name %> </h1>**

The <%= ... %> is a method link tag, in Rails 'speak'.

It is translated in this case to be: Get the value of the instance variable @name and substitute it here please.

If we view our page again we will just get **Hello**

The controller has nowhere to get the data from so the instance variable is empty!

If we add a little more Rails code to our index page, we can turn it into a form and allow the user to enter his/her name and the controller will automatically catch this for us and display it on the page.

We need to add:

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

This is the Rails code that creates a <form> entry in our html and sets the action to call the index page again.

<p>

Enter Employee Name:

<%= text\_field\_tag 'name' , @name %>

</p>

This code will cause Rails to generate a text box with an identifier of 'visitor\_name' and if the instance variable contains a value insert this into the box or leave it blank if there is no value.

<%= submit\_tag 'Go!' %>

This generate the usual submit button on the page with the text 'Go!' on it.

Finally

<% end %>

This is a plain Rails tag signifying the end of the code block for the **'do'** in the **form\_tag**.

If we re-run the page in the browser, we can see the results of our changes.

## Actionviews and 'DRY'

One of the design philosophies of Rails applications is DRY – Do Not Repeat Yourself.

There are many scenarios where for example you may have the same form code used in several places; it would be counter to DRY to keep re-writing the same code.

Rails allow you define the code once and store it in a file within the view and 'include' this into other views when needed.

The file name starts with and underscore and then follows the normal naming convention.

For example we can create our form code and store it in a file called:

**\_links.html.erb**

In each view where we need the form code to be used we add the following rails tag

<%= render 'links' %>

**Note**: the leading underscore is dropped from the file name, and we only specify the file name not the extension as well, on the call.

This causes Rails to 'insert' the code from the file at this point in the view.

## Page linking

One of the most important aspects of your application is the navigation through it.

It would be an extremely tedious, if not impossible, job to create and maintain the necessary HTML <a> anchor tags to support the 'site flow'.

Rails, 'to the rescue', has a link\_tag that will do the job for you.

The link\_tag, like most of Rails, is extremely versatile.

It supports simple linking, it can pass parameters, there is even the ability to make it conditional based as well.

The simplest of links is:

link\_to(body, url)

Where **body** is the displayed name for the link and the **url** becomes the target.

This is where Rails routing also come into play, you have to understand what routes you have.

Consider the following small 'snippets' of a rails application.

From the routes.rb file

resources :employees

Running rake routes gives:

employees GET /employees(.:format) employees#index

POST /employees(.:format) employees#create

new\_employee GET /employees/new(.:format) employees#new

edit\_employee GET /employees/:id/edit(.:format) employees#edit

employee GET /employees/:id(.:format) employees#show

PATCH /employees/:id(.:format) employees#update

PUT /employees/:id(.:format) employees#update

DELETE /employees/:id(.:format) employees#destroy

The associated controller needs to contain a method for each of the routes:

def index

end

def show

end

def new

end

def edit

end

def create

end

the apps\views directory contain a file for each of the views defined in the controller.

If we create a file in apps\views called **\_links.html.erb** and add the following to it:

<%= link\_to 'New Employee', new\_employee\_path %>

<%= link\_to 'Show', employee %>

And the in each or our views we add:

<%= render 'links' %>

We will get a set of link on every page generated by the view that will allow for navigation around our 'fictitious' site.

When you create a Rails application using scaffold it will automatically create certain links in the views for you.

We will see this in chapter 7 later.

You can also put 'real' web references into you links as well as shown below:

<%= link\_to 'Work', "http://stayahead.com" %>

## Form helpers

One of the features of rails is the considerable number of 'helpers' available.

The purpose of the helpers is to save the developer time in adding features to the views.

For example, using the model relationship describer earlier between employees and departments we need to create a view that allows a new employee to be added, this will involve 3 input fields: employee name, employee extension and employee department.

There are several form helpers available.

The example code below uses the **form\_for** helper to build the overall form itself.

Form the form perspective we need to be able to select the department from a list derived from the department table, a classic HTML select object. We need to store the department as an integer, as defined in the employee migration file but we would like to show the list of possibilities by using the department names.

If we created the select drop down manually then every time we added a new department to the company we would have to redefine the select code.

Rails provide a **collection\_select** helper to automate the building of a select drop down, as this is dynamic any new department will be added automatically.

The basic syntax for the select drop down for our example is shown below:

**<%= f.collection\_select(:department\_id, Department.all, :id, :name, prompt: true )%>**

**Where:**

|  |  |
| --- | --- |
| **:department\_id** | **Target column in table** |
| **Department.all** | **Data source(Table)** |
| **:id** | **Value to store from the select** |
| **:name** | **Display value in select drop-down** |
| **prompt: true** | **Add select title to top of drop-down** |

Additionally we want to enforce the data input not to allow empty fields.

By adding the **:required => true** parameter to the input rails will create the HTML to ensure that the data input field is not blank

### The completed form example

<%= form\_for(employee) do |f| %>

<div class="field">

<%= f.label :name %>

<%= f.text\_area :name, :required => true %>

</div>

<div class="field">

<%= f.label :phone %>

<%= f.text\_area :phone, :required => true %>

</div>

<div class="field">

<%= f.label :department\_id %>

<%= f.collection\_select(:department\_id, Department.all, :id, :name, prompt: true )%>

</div>

<div class="actions">

<%= f.submit %>

</div>

<% end %>

### The resultant HTML

<html>

.

. [output truncated]

.

<body>

<h1>New Employee</h1>

<form class="new\_employee" id="new\_employee" action="/employees" accept-charset="UTF-8" method="post"><input name="utf8" type="hidden" value="&#x2713;" /><input type="hidden" name="authenticity\_token" value="70zSQS0uEr+l4LnzjRRa61wFHXozsJnmEEeG1jPXU3F84+NWaSCEoeqf9PT3uynBjiNWupEP42+Wo+kQi2wcqw==" />

<div class="field">

<label for="employee\_name">Name</label>

<textarea **required="required"** name="employee[name]" id="employee\_name">

</textarea>

</div>

<div class="field">

<label for="employee\_phone">Phone</label>

<textarea **required="required"** name="employee[phone]" id="employee\_phone">

</textarea>

</div>

<div class="field">

<label for="employee\_department\_id">Department</label>

**<select name="employee[department\_id]" id="employee\_department\_id"><option value="">Please select</option>**

**<option value="1">HR</option>**

**<option value="2">Production</option></select>**

</div>

<div class="actions">

<input type="submit" name="commit" value="Create Employee" data-disable-with="Create Employee" />

</div>

</form>

<a href="/employees">Back</a>

</body>

</html>

## Appearance

With the standard concept in rails of 'DRY', you can apply the same approach to the appearance and layout of your site.

The code below is a simple example of this.

To apply a set of styles across the application, one method would be to define the styles in the global stylesheet held in **app/assets/stylesheets/application.css**

For example:

.footer {

position: fixed;

left: 0;

bottom: 0;

width: 100%;

background-color: PowderBlue;

color: white;

text-align: center;

}

body {

background-color: #88918d;

}

H1 {

color: #ffff66

}

Similarly, to provide global layouts throughout the site you can define your own elements in **app/views/layouts/application.html.erb**

For example:

<!DOCTYPE html>

<html>

<head>

<title>Company</title>

<%= csrf\_meta\_tags %>

<%= stylesheet\_link\_tag 'application', media: 'all', 'data-turbolinks-track': 'reload' %>

<%= javascript\_include\_tag 'application', 'data-turbolinks-track': 'reload' %>

</head>

<body>

<%= yield %>

</body>

**<div class="footer">**

**<footer>**

**<ul>**

**<li><%= link\_to 'Show all Employees', employees\_path %></li>**

**<li><%= link\_to 'Create new Employee', new\_employee\_path %></li>**

**<li><%= link\_to 'Show all Departments', departments\_path %></li>**

**<li><%= link\_to 'Create new Department', new\_department\_path%></li>**

**<li><%= link\_to 'home', root\_path%></li>**

**</ul>**

**</footer>**

**</div>**

</html>

There any many ways to provide layout and styling to your site, you are encouraged to research this via the web or Rails documentation.