CHAPTER 6

# Rails Project

# Creating a basic rails application

In this section we are going to create a simple Rails application based on a Guest Book, as nearly every rails tutorial uses this!

## Creating the basic application

Using the rails command, from our base directory - **/sites** - we can generate the new application by entering the following.

rails new guestbook

This will now create the rails file structure, as was shown in chapter 1, below **/sites** in a sub-directory called **guestbook**

All the work needed to complete the application is done from within the **/sites/guestbook** directory.

## Creating the database

Nearly every Rails application will interact with a database.

The database to use is specified in a configuration file, config/database.yml.

If you examine this file in our new Rails application, you’ll see a default database configured to use SQLite3.

The file contains sections for three different environments in which Rails can run by default.

* The **development** environment is used on your development/local computer as you interact manually with the application.
* The **test** environment is used when running automated tests.
* The **production** environment is used when you deploy your application for the world to use.

Create the database using the following rake command, rake is similar to the 'make' facility used by programmers etc.

rake db:create

## Starting up the Web Server

In order to 'view' our new application we need a web server to deliver it to our browser.

Rails comes with a built-in web server called puma and this is started using the rails command.

As before in our training examples we are going to run Puma listening to port 80, so we need to modify the config/puma.rb file to bind the server to port 80.

Now in a separate terminal window cd to **/sites/guestbook** directory and run:

rails server (this can be abbreviated to rails s)

We can now test our application by pointing a browser to **http://localhost**

If the application and server are running correctly, we will get the rails default page.



## Creating the controller

We need to create a basic controller for the application, again rails can do this for use by executing the following rails command.

**rails generate controller entries**

The **rails generate** command can be abbreviated to **rails g**, and this is how we will use it from now on.

We now have a basic controller in place. Although at this stage it has no 'real' functionality associated with it.

The controller is located in

***app/controllers/entries\_controller.rb***

We need to add a method to the controller.

Using a text editor open the ***entries\_controller.rb*** and add the following methodto the class already defined.

**def sign\_in**

**@name = params[:visitor\_name]**

**end**

This method will get the visitor\_name parameter from the request header and put it into the instance variable **@name**.

## Creating the view

In order to send data to our application we need a view that will allow a user to input his/her name.

Using the text editor create a file **app/views/entries/sign\_in.html.erb** and insert the following code.

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

**<%= form\_tag :action => 'sign\_in' do %>**

**<p> Enter your name:**

**<%= text\_field\_tag 'visitor\_name' , @name %> </p>**

**<%= submit\_tag 'Sign In' %>**

**<% end %>**

**Note:** We could have created both a basic controller and view outline at the same time by using **rails g controller entries sign\_in**

## Creating the routes

The final step for the first part is to create the routes for the application. These are held in a file **config/routes.rb**.

get 'entries/sign\_in'

### Testing part 1

Now our application should start to be able to do something.

Point your browser to:

**http://localhost/entries/sign\_in**

And you should now see the form. You can now enter your Name and press the submit button.

## Creating the model

In order for our new application to store the entered names in the database we need to create a model to deal with this.

Again rails can create out model outline for us with a simple command

rails g model entry

This will create two files:

* **app/models/entry.rb**
* **db/migrate/<timestamp>\_create\_entries.rb**

The entry.rb file will contain the logic for dealing with the data and the <timestamp>\_create\_entries.rb contains the outline for the database table

The <timestamp> will be replaced by the 'real' timestamp when the file was created. This is a mechanism that rails can use to keep track of files that are auto generated and that can be used for rollback etc.

Our first step is to alter the <timestamp>\_create\_entries.rb and add a line that will create a column in the created table to store our names.

From:

class CreateEntries < ActiveRecord::Migration

def change

create\_table :entries do |t|

t.timestamps

end

end

end

To:

class CreateEntries < ActiveRecord::Migration

def change

create\_table :entries do |t|

t.string :name

t.timestamps

end

end

end

We now need to run the rake command again to create the table.

rake db:migrate

The next step is to link the controller and the model together. This is achieved by modifying the controller for the application.

Again using a text editor open the ***entries\_controller.rb*** and modify the methodin the class.

From:

def sign\_in

@name = params[:visitor\_name]

end

To:

def sign\_in

@name = params[:visitor\_name]

@entry = Entry.create({:name => @name})

end

You may want to avoid blank names going into the table if someone just hits the submit without entering a name.

Just modify the view code to add simple testing, ideally this is something the model should do not the controller.

def sign\_in

@name = params[:visitor\_name]

unless @name.blank?

@entry = Entry.create({:name => @name})

end

end

We also have to add a route so that rails knows how to deal with the post request generated when the submit button is pressed. We are going to call the same view, so we need to add a post route in the config/routes.rb file as follows:

Rails.application.routes.draw do

get 'entries/sign\_in'

**post 'entries/sign\_in'**

# For details on the DSL available within this file, see http://guides.rubyonrails.org/routing.html

End

Running:

rails routes

Now gives us:

Prefix Verb URI Pattern Controller#Action

entries\_sign\_in GET /entries/sign\_in(.:format) entries#sign\_in

POST /entries/sign\_in(.:format) entries#sign\_in

That is the input side dealt with.

Now we need to add some further functionality to get our data back from our table.

The first part is to modify the controller to do this by modifying the method as follows:

def sign\_in

@name = params[:visitor\_name]

unless @name.blank?

@entry = Entry.create({:name => @name})

end

@entries = Entry.all

end

Now we have to modify the view to display the information.

Currently:

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

<%= form\_tag :action => 'sign\_in' do %>

<p> Enter your name:

<%= text\_field\_tag 'visitor\_name' , @name %> </p>

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

<% end %>

To:

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

<%= form\_tag :action => 'sign\_in' do %>

<p> Enter your name:

<%= text\_field\_tag 'visitor\_name' , @name %> </p>

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

<% end %>

<p>Previous Visitors:</p>

<ul>

<% @entries.each do |entry| %>

<li><%= entry.name %> </li>

<% end %>

</ul>

Congratulations your first rails application is now complete – enjoy!!!!

## Workshop 1

Cary out the following steps:

1. In the /sites directory run the **rails new** command to create a basic rails application called **company**
2. Using the rails generator create a model named employee and a controller called employees
3. Edit the generated index page to allow the user to enter and store the following details for each employee:
   1. First Name
   2. Surname
   3. Job Title
   4. Extension

1. Use the create view to:  
   1. Display whether the new record was added or not
2. Show all the current employees

# Workshop 1 Solution

Cary out the following steps:

1. In the /home/user1/bin directory run the **rails new** command to create a basic rails application called **company**
2. Using the rails generator create a model and a controller called employee
3. Edit the generated index page to allow the user to enter and store the following details for each employee:
   1. First Name
   2. Surname
   3. Job Title
   4. Extension
4. Display whether the new record was added or not
5. Show all the current employees

Assuming project home of **/sites**

Create the rails project:

rails new company

cd into company

Create the model and controller:

rails g model employees fname:text sname:text title:text extn:text

rails g controller employees index

Create database and table:

rake db:create

rake db:migrate

Modify routes to automate route creation

vi config/routes.rb

replace get entry with:

resources :employees

Test routes with:

rake routes

## Controller

Create the entries for index, new, show and create actions.

Include white list and validation for user data

vi app/controllers/employees\_controller.rb

class EmployeesController < ApplicationController

before\_action :set\_employee, only: [:show ]

def index

@employees = Employee.all

end

def new

@employee = Employee.new

end

def show

end

def create

@employee = Employee.new(employee\_params)

respond\_to do |format|

if @employee.save

format.html { redirect\_to @employee, notice: 'Employee was successfully created.' }

else

format.html { render :new }

end

end

end

private

def employee\_params

params.require(:employee).permit(:fname, :sname, :title, :extn)

end

def set\_employee

@employee = Employee.find(params[:id])

end

end

## Views

Create the views for the index, new and show actions, to support the 'DRY' concept the solution includes a \_form.html.erb file as an example.

### index.html.erb

<h1>Employees</h1>

<table>

<thead>

<tr>

<th>First Name</th>

<th>Surname</th>

<th>Title</th>

<th>Extension</th>

<th colspan="4"></th>

</tr>

</thead>

<tbody>

<% @employees.each do |e| %>

<tr>

<td><%= e.fname %></td>

<td><%= e.sname %></td>

<td><%= e.title %></td>

<td><%= e.extn %></td>

</tr>

<% end %>

</tbody>

</table>

<br>

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

### new.html.erb

<h1>New Employee</h1>

<%= render 'form', employee: @employee %>

<%= link\_to 'Back', employees\_path %>

### \_form.html.erb

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

<% if employee.errors.any? %>

<div id="error\_explanation">

<h2><%= pluralize(employee.errors.count, "error") %> prohibited this employee from being saved:</h2>

<ul>

<% employee.errors.full\_messages.each do |message| %>

<li><%= message %></li>

<% end %>

</ul>

</div>

<% end %>

<div class="field">

<%= f.label :fname %>

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

</div>

<div class="field">

<%= f.label :sname %>

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

</div>

<div class="field">

<%= f.label :title %>

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

</div>

<div class="field">

<%= f.label :entn %>

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

</div>

<div class="actions">

<%= f.submit %>

</div>

<% end %>

### show.html.erb

<p id="notice"><%= notice %></p>

<p>

<strong>First Name:</strong>

<%= @employee.fname %>

</p>

<p>

<strong>Surname:</strong>

<%= @employee.sname %>

</p>

<p>

<strong>Title:</strong>

<%= @employee.title %>

</p>

<p>

<strong>Extn:</strong>

<%= @employee.extn %>

</p>

<%= link\_to 'Back', employees\_path %>