Ruby on Rails – Installation

http://www.tutorialspoint.com/ruby-on-rails/rails-installation.htm

This tutorial will guide you to set up a private Ruby on Rails environment in the “daw” server.

Step 0: Login to the daw server

```
a999999@australia:~$ ssh a999999@10.10.23.183
```

Step 1: Install rbenv

```
a999999@daw:~$ git clone git://github.com/sstephenson/rbenv.git .rbenv
a999999@daw:~$echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bash_profile
a999999@daw:~$echo 'eval "$(rbenv init -)"' >> ~/.bash_profile


a999999@daw:~$git clone git://github.com/sstephenson/ruby-build.git
~/.rbenv/plugins/ruby-build
a999999@daw:~$echo 'export PATH="$HOME/.rbenv/plugins/ruby-build/bin:$PATH"' >>
~/.bash_profile

a999999@daw:~$echo "export PATH=~/.gem/ruby/2.2.0/bin:\$PATH" >> ~/.bash_profile
a999999@daw:~$echo "export GEM_HOME=~/.gem/ruby/2.2.0/gems" >> ~/.bash_profile
a999999@daw:~$echo "export GEM_PATH=~/.gem/ruby/2.2.0" >> ~/.bash_profile

a999999@daw:~$source ~/.bash_profile
```

Step 2: Install Ruby

Bofore install Ruby, First determine which version of ruby that you want to install. We will install Ruby 2.2.3. Use the following command for installing Ruby 2.2.3.

```
a999999@daw:~$ rbenv install --list
a999999@daw:~$ rbenv install -v 2.2.3
a999999@daw:~$ rbenv global 2.2.3
a999999@daw:~$ ruby -v


a999999@daw:~$ echo "gem: --no-document" > ~/.gemrc
a999999@daw:~$ echo "gem: --user-install" >> ~/.gemrc
```

Step 3: Install Rails

The following command for installing rails of 4.2.4 version

```
a999999@daw:~$ git clone git://github.com/jmatbastos/ruby_on_rails.git .gem
a999999@daw:~$cd ~/.gem/ruby/2.2.0/bin
```

In the following command replace “a12345” with your login

```
a999999@daw:~/.gem/ruby/2.2.0/bin$ for file in `ls` ;  do cat $file | sed
s/a999999/a12345/ > ${file}.tmp ; mv ${file}.tmp $file ; chmod a+x $file; done

a999999@daw:~/.gem/ruby/2.2.0/bin$ cd

a999999@daw:~$ gem install bundler
a999999@daw:~$ gem install --version '>=5.1' minitest
```

Use the following command to make rails executable available.

```
a999999@daw:~$ rbenv rehash
```

Use the following command for checking the rails version.

```
a999999@daw:~$ rails –v
```

Step 4: Installation Verification

You can verify if everything is setup according to your requirements or not. Use the following command to create a demo project.

```
a999999@daw:~$ cd public_html
a999999@daw:~/public_html$ rails new demo

a999999@daw:~/public_html$ cd demo
```

In the following commands replace “12345” with your student number

```
a999999@daw:~/public_html/demo$ rails server -p 12345 –b 10.10.23.183
```

Now open your browser and type the following address text box.

http://10.10.23.183:12345

Ruby on Rails – library site

http://www.tutorialspoint.com/ruby-on-rails/rails-installation.htm

Step 1: New project "library"

Use the following command to create a library project.

```
a999999@daw:~$ cd public_html
a999999@daw:~/public_html$ rails new library

a999999@daw:~/public_html$ cd library
```

In the following commands replace "12345" with your student number

```
a999999@daw:~/public_html/library$ rails server -p 12345 –b 10.10.23.183
```

Now open your browser and type the following address text box.

```
http://10.10.23.183:12345
```

Step 2: Configure the access to the mysql database

At this point, you need to let Rails know about the user name and password for the databases. You do this in the file **database.yml**, available in the **library/config** subdirectory of Rails Application you created. This file has live configuration sections for MySQL databases. In each of the sections you use, you need to change the username and password lines to reflect the permissions on the databases you've created.

When you finish, it should look something like this (change a12345 with your login)

```
development:
   adapter: mysql
   database: db_a12345
   username: a12345
   password: [password]
   host: 10.10.23.13
```

Step 3: Translating a Domain Model into SQL

Translating a domain model into SQL is generally straight forward, as long as you remember that you have to write Rails-friendly SQL. In practical terms, you have to follow certain rules:

- Each entity (such as book) gets a table in the database named after it, but in the plural (books).
- Each such entity-matching table has a field called *id*, which contains a unique integer for each record inserted into the table.
- Given entity x and entity y, if entity y belongs to entity x, then table y has a field called x_id.

- The bulk of the fields in any table store the values for that entity's simple properties (anything that's a number or a string).

```
a999999@daw:~/public_html/library$ rails generate model Book

a999999@daw:~/public_html/library$ rails generate model Subject
```

Step 4: Creating Associations between Models

When you have more than one model in your rails application, you would need to create connection between those models. You can do this via associations. Active Record supports three types of associations −

- **one-to-one** − A one-to-one relationship exists when one item has exactly one of another item. For example, a person has exactly one birthday or a dog has exactly one owner.
- **one-to-many** − A one-to-many relationship exists when a single object can be a member of many other objects. For instance, one subject can have many books.
- **many-to-many** − A many-to-many relationship exists when the first object is related to one or more of a second object, and the second object is related to one or many of the first object.

You indicate these associations by adding declarations to your models: has_one, has_many, belongs_to, and has_and_belongs_to_many.

To do so, modify app/models/book.rb and app/models/subject.rb to look like this −

```
class Book < ActiveRecord::Base
   belongs_to :subject
end
```

```
class Subject < ActiveRecord::Base
   has_many :books
end
```

Step 5: Implementing Validations on Models

The implementation of validations is done in a Rails model. The data you are entering into the database is defined in the actual Rails model, so it only makes sense to define what valid data entails in the same location.

The validations are −

- The value of title field should not be NULL.
- The value of price field should be numeric.

Open **book.rb** in the **app/model** subdiractory and put the following validations

Step 6: Create the Migrations

We will create two migrations corresponding to our two tables − ***books and subjects***.

Books migration should be as follows −

```
a999999@daw:~/public_html/library$ rails generate migration books
a999999@daw:~/public_html/library$ rails generate migration subjects
```

Step 7: Create the tables

Go to db/migrate subdirectory of your application and edit each file one by one using any simple text editor.

Modify 20151234567890_books.rb as follows −

```
class Books < ActiveRecord::Migration

   def self.up
      create_table :books do |t|
         t.column :title, :string, :limit => 32, :null => false
         t.column :price, :float
         t.column :subject_id, :integer
         t.column :description, :text
         t.column :created_at, :timestamp
      end
   end

   def self.down
      drop_table :books
   end
end
```

Modify 20151234567890_subjects.rb as follows −

```
class Subjects < ActiveRecord::Migration
   def self.up

      create_table :subjects do |t|
         t.column :name, :string
      end

      Subject.create :name => "Physics"
      Subject.create :name => "Mathematics"
      Subject.create :name => "Chemistry"
      Subject.create :name => "Psychology"
      Subject.create :name => "Geography"
   end

   def self.down
      drop_table :subjects
   end
end
```

Now create the tables

```
a999999@daw:~/public_html/library$ export RAILS_ENV=development

a999999@daw:~/public_html/library$ rake db:migrate
```

Step 8: Generate the controllers

```
a999999@daw:~/public_html/library$ rails generate controller Books
```

This command accomplishes several tasks, of which the following are relevant here −

It creates a file called **app/controllers/books_controller.rb**

Implementing the list Method

The list method gives you a list of all the books in the database. This functionality will be achieved by the following lines of code. Edit the following lines in books_controller.rb file.

```
def list
  @books = Book.all
end
```

The *@books = Book.all* line in the list method tells Rails to search the books table and store each row it finds in the @books instance object.

Implementing the show Method

The show method displays only further details on a single book. This functionality will be achieved by the following lines of code.

```
def show
   @book = Book.find(params[:id])
end
```

The show method's *@book = Book.find(params[:id])* line tells Rails to find only the book that has the id defined in params[:id].

The params object is a container that enables you to pass values between method calls. For example, when you're on the page called by the list method, you can click a link for a specific book, and it passes the id of that book via the params object so that show can find the specific book.

Implementing the new Method

The new method lets Rails know that you will create a new object. So just add the following code in this method.

```
def new
   @book = Book.new
   @subjects = Subject.all
end
```

The above method will be called when you will display a page to the user to take user input. Here second line grabs all the subjects from the database and puts them in an array called @subjects.

Implementing the create Method

Once you take user input using HTML form, it is time to create a record into the database. To achieve this, edit the create method in the book_controller.rb to match the following −

```
def create
   @book = Book.new(book_params)

   if @book.save
      redirect_to :action => 'list'
   else
      @subjects = Subject.all
```

```
    render :action => 'new'
  end


end


def book_params
  params.require(:books).permit(:title, :price, :subject_id, :description)
end
```

The first line creates a new instance variable called @book that holds a Book object built from the data, the user submitted. The **book_params** method is used to collect all the fields from object **:books**. The data was passed from the new method to create using the params object.

The next line is a conditional statement that redirects the user to the **list** method if the object saves correctly to the database. If it doesn't save, the user is sent back to the new method. The redirect_to method is similar to performing a meta refresh on a web page: it automatically forwards you to your destination without any user interaction.

Then *@subjects = Subject.all* is required in case it does not save data successfully and it becomes similar case as with new option.

Implementing the edit Method

The edit method looks nearly identical to the show method. Both methods are used to retrieve a single object based on its id and display it on a page. The only difference is that the show method is not editable.

```
def edit
  @book = Book.find(params[:id])
  @subjects = Subject.all
end
```

This method will be called to display data on the screen to be modified by the user. The second line grabs all the subjects from the database and puts them in an array called @subjects.

Implementing the update Method

This method will be called after the edit method, when the user modifies a data and wants to update the changes into the database. The update method is similar to the create method and will be used to update existing books in the database.

```
def update
```

```
  @book = Book.find(params[:id])

  if @book.update_attributes(book_param)
    redirect_to :action => 'show', :id => @book
  else
    @subjects = Subject.all
    render :action => 'edit'
  end

end

def book_param
  params.require(:book).permit(:title, :price, :subject_id, :description)
end
```

The update_attributes method is similar to the save method used by create but instead of creating a new row in the database, it overwrites the attributes of the existing row.

Then *@subjects = Subject.all* line is required in case it does not save the data successfully, then it becomes similar to edit option.

Implementing the delete Method

If you want to delete a record from the database then you will use this method. Implement this method as follows.

```
def delete
  Book.find(params[:id]).destroy
  redirect_to :action => 'list'
end
```

The first line finds the classified based on the parameter passed via the params object and then deletes it using the destroy method. The second line redirects the user to the list method using a redirect_to call.

Additional Methods to Display Subjects

Assume you want to give a facility to your users to browse all the books based on a given subject. So, you can create a method inside book_controller.rb to display all the subjects. Assume the method name is **show_subjects** −

```
def show_subjects
  @subject = Subject.find(params[:id])
end
```

Finally your **books_controller.rb** file will look as follows −

```
class BooksController < ApplicationController

   def list
      @books = Book.all
   end

   def show
      @book = Book.find(params[:id])
   end

   def new
      @book = Book.new
      @subjects = Subject.all
   end

   def book_params
      params.require(:books).permit(:title, :price, :subject_id,
:description)
   end

   def create
      @book = Book.new(book_params)

      if @book.save
         redirect_to :action => 'list'
      else
         @subjects = Subject.all
         render :action => 'new'
      end
   end
```

```
   def edit
      @book = Book.find(params[:id])
      @subjects = Subject.all
   end

   def book_param
      params.require(:book).permit(:title, :price, :subject_id,
:description)
   end

   def update
      @book = Book.find(params[:id])

      if @book.update_attributes(book_param)
         redirect_to :action => 'show', :id => @book
      else
         @subjects = Subject.all
         render :action => 'edit'
      end
   end

   def delete
      Book.find(params[:id]).destroy
      redirect_to :action => 'list'
   end

   def show_subjects
      @subject = Subject.find(params[:id])
   end

end
```

Step 9: Generate routes

Open routes.rb file in library/config/ directory and edit it with the following content.

```
Rails.application.routes.draw do
   root'books#list'
   get 'books/list'
```

```
   get 'books/new'
   post 'books/create'
   patch 'books/update'
   get 'books/list'
   get 'books/show'
   get 'books/edit'
   get 'books/delete'
   get 'books/update'
   get 'books/show_subjects'
end
```

The routes.rb file defines the actions available in the applications and the type of action such as get, post, and patch.

Step 10: Create views

Now, display the actual content. Let us put the following code into list.html.erb.

```
<% if @books.blank? %>
<p>There are not any books currently in the system.</p>
<% else %>
<p>These are the current books in our system</p>

<ul id = "books">
   <% @books.each do |c| %>
   <li>
    <%= link_to c.title, {:action => 'show', :id => c.id} -%>
    <b> <%= link_to “Edit”, {:action => ’edit’, :id => c.id} %></b>
   <b> <%= link_to "Delete", {:action => 'delete', :id => c.id},
      :confirm => "Are you sure you want to delete this item?" %></b>
   </li>
   <% end %>
</ul>

<% end %>
<p><%= link_to "Add new Book", {:action => 'new' }%></p>
```

Create a file called new.html.erb using your favorite text editor and save it to app/views/book. Add the following code to the new.html.erb file.

```
<h1>Add new book</h1>
```

```
<%= form_tag :action => 'create' do %>
<p><label for = "book_title">Title</label>:

<%= text_field 'books', 'title' %></p>
<p><label for = "book_price">Price</label>:

<%= text_field 'books', 'price' %></p>
<p><label for = "book_subject_id">Subject</label>:

<%= collection_select(:books, :subject_id, @subjects, :id, :name, prompt: true)
%></p>
<p><label for = "book_description">Description</label><br/>

<%= text_area 'books', 'description' %></p>
<%= submit_tag "Create" %>

<% end -%>
<%= link_to 'Back', {:action => 'list'} %>
```

Create a show.html.erb file under app/views/book and populate it with the following code −

```
<h1><%= @book.title %></h1>

<p>
   <strong>Price: </strong> $<%= @book.price %><br />
   <strong>Subject :</strong> <%= link_to @book.subject.name, :action =>
"show_subjects", :id => @book.subject.id %>
 <br />
   <strong>Created Date:</strong> <%= @book.created_at %><br />
</p>

<p><%= @book.description %></p>

<hr />

<%= link_to 'Back', {:action => 'list'} %>
```

Create a new file called edit.html.erb and save it in app/views/book. Populate it with the following code −

```
<h1>Edit Book Detail</h1>

<%= form_for @book, :url =>{:action => "update", :id =>@book} do |f| %>

<p>Title: <%= f.text_field 'title' %></p>
<p>Price: <%= f.text_field  'price' %></p>
<p>Subject: <%= f.collection_select :subject_id, Subject.all, :id, :name %></p>
<p>Description<br/>

<%= f.text_area 'description' %></p>
<%= f.submit "Save changes" %>
<% end %>

<%= link_to 'Back', {:action => 'list' } %>
```

Create a new file, show_subjects.html.erb, in the app/views/book directory and add the following code to it −

```
<h1><%= @subject.name -%></h1>

<ul>
   <% @subject.books.each do |c| %>
   <li><%= link_to c.title, :action => "show", :id => c.id -%></li>
   <% end %>
</ul>
```

Step 11: Create layout

Add a new file called standard.html.erb to app/views/layouts. You let the controllers know what template to use by the name of the file, so following a same naming scheme is advised.

Add the following code to the new standard.html.erb file and save your changes −

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns = "http://www.w3.org/1999/xhtml">

   <head>
      <meta http-equiv = "Content-Type" content = "text/html; charset = iso-8859-1"
/>
      <meta http-equiv = "Content-Language" content = "en-us" />
      <title>Library Info System</title>
      <%= stylesheet_link_tag "style" %>
   </head>

   <body id = "library">
      <div id = "container">

         <div id = "header">
            <h1>Library Info System</h1>
            <h3>Library powered by Ruby on Rails</h3>
         </div>

         <div id = "content">
            <%= yield -%>
         </div>

         <div id = "sidebar"></div>

      </div>
   </body>

</html>
```

Now open **book_controller.rb** and add the following line just below the first line −

```
class BookController < ApplicationController
layout 'standard'
def list
```

```
@books = Book.all
end
.....................
```

Step 12: Adding Style Sheet

Till now, we have not created any style sheet, so Rails is using the default style sheet. Now let's create a new file called style.css and save it in /public/stylesheets. Add the following code to this file.

```
body {
   font-family: Helvetica, Geneva, Arial, sans-serif;
   font-size: small;
   font-color: #000;
   background-color: #fff;
}

a:link, a:active, a:visited {
   color: #CD0000;
}

input {
   margin-bottom: 5px;
}

p {
   line-height: 150%;
}

div#container {
   width: 760px;
   margin: 0 auto;
}

div#header {
   text-align: center;
   padding-bottom: 15px;
```

```
}

div#content {
   float: left;
   width: 450px;
   padding: 10px;
}

div#content h3 {
   margin-top: 15px;
}

ul#books {
   list-style-type: none;
}

ul#books li {
   line-height: 140%;
}

div#sidebar {
   width: 200px;
   margin-left: 480px;
}

ul#subjects {
   width: 700px;
   text-align: center;
   padding: 5px;
   background-color: #ececec;
   border: 1px solid #ccc;
   margin-bottom: 20px;
}

ul#subjects li {
   display: inline;
```

```
   padding-left: 5px;
}
```