Rails application layout

In this chapter, we will learn how to use Bootstrap with Cascade Style Sheets (CSS) to format the web application. Bootstrap is the frame work provided by the twitter to decorate the web page.

Create a new application

let's start this handout by creating a new application in rails.

```
$ rails new bootstrapapp -d postgresql
$ cd bootstrapapp
$ rake db:create
```

Generating static pages

Lets generate some static pages to apply the bootstrap CSS on it.

```
$ rails generate controller StaticPages home help
```

By writing home and help with StaticPages controller will do the following automatically

- Will enter the two actions *home* and *help* in the static_pages controller.
- Will make entry for the home and help page in the config\routes.rb file.
- Will create the two view pages *home.html.erb* and *help,html.erb*.

Custom static pages

AS the above code generate the view, controller and path in routes automatically. Suppose we want to add one more static pages. As the controller is already created, we have to manually add a static page, in this case *about*.

Entry in the routes

Let's make an entry for the about page in the *routes.rb* file.

- Entry in the routes file.

  config/routes.rb

  ```
  get 'static_pages/about'
  ```

- Adding the **about** action in the controller

  static_pages_controller.rb

  ```
  def about
  end
  ```

- View page for the about.

app/views/static_pages/about.html.erb

```
        <h1> About </h1>
```

A slightly dynamic pages

Layouts in rails

In this section we will learn how to set the layouts for the web application using rails.

Site Navigation

Let's add links and styles to the application. *Application.html.erb* is the common place to add the structure with HTML. It also include additional division, some CSS classes, and the start of the page navigation.

application.html.erb

```
  <body>
    <header class="navbar navbar-fixed-top navbar-inverse">
      <div class="container">
        <%= link_to "sample app","#",id: "logo" %>
        <nav>
          <ul class="nav navbar-nav navbar-right">
            <li><%= link_to "Home","#" %></li>
            <li><%= link_to "Help","#" %></li>
            <li><%= link_to "Log in","#" %></li>
          </ul>
        </nav>
      </div>
    </header>

    <div class="container">
      <%= yield %>
    </div>
  </body>
```

The home page with a link to the sign-up page.

home.html.erb

```
    <div class="center jumbotron">
      <h1>Welcome to the Sample App</h1>
      <h2>
        This is the home page for the
        <a href="http://www.railstutorial.org/">Ruby on Rails Tutorial</a>
        sample application.
      </h2>
    <%= link_to "Sign up now!","#",class: "btn btn-lg
        btn-primary" %>
    </div>

    <%= link_to image_tag("rails.png",alt: "Rails logo"),
'http://rubyonrails.org/' %>
```

Bootstrap and custom CSS

This section will work with bootstrap and creating custom CSS files. Lets enable the rails application to use bootstrap.

1. Adding the *bootstrap-sass* gem into the **Gemfile**.

   Gemfile

   ```
   gem 'bootstrap-sass'
   ```

2. to install the bootstrap gem use

   ```
   $ bundle install
   ```

3. Add the *custom.css.scss* into app/assets/stylesheets and add the following lines into the custom file.

   app/assets/stylesheets/custom.css.scss

   ```
   @import "bootstrap-sprockets";
   @import "bootstrap";
   ```

   and restart the server.

Adding CSS for some universal styling applying to all pages.

app/assets/stylesheets/custom.css.scss

```
/* universal */

  body {
    padding-top: 60px;
  }

  section {
    overflow: auto;
  }

  textarea {
    resize: vertical;
  }

  .center {
    text-align: center;
  }

  .center h1 {
    margin-bottom: 10px;
  }
```

Lets add some custom rules for the appearance of the text on our site.

app/assets/stylesheets/custom.css.scss

```
/* typography */

h1,h2,h3,h4,h5,h6 {
  line-height: 1;
}

h1 {
  font-size: 3em;
  letter-spacing: -2px;
  margin-bottom: 30px;
  text-align: center;
}

h2 {
  font-size: 1.2em;
  letter-spacing: -1px;
  margin-bottom: 30px;
  text-align: center;
  font-weight: normal;
  color: #777;
}

p {
  font-size: 1.1em;
  line-height: 1.7em;
}
```

Partials

Lets include some partials to have a clean code in the application.html.erb.

app/views/layouts/_header.html.erb

```
<header class="navbar navbar-fixed-top navbar-inverse">
  <div class="container">
    <%= link_to "sample app","#",id: "logo" %>
    <nav>
      <ul class="nav navbar-nav navbar-right">
        <li><%= link_to "Home","#" %></li>
        <li><%= link_to "Help","#" %></li>
        <li><%= link_to "Log in","#" %></li>
      </ul>
    </nav>
  </div>
</header>
```

and the footer partial.

app/views/layouts/_footer.html.erb

```
<footer class="footer">
  <small>
    The <a href="http://www.railstutorial.org/">Ruby on Rails Tutorial</a>
    by <a href="http://www.michaelhartl.com/">Michael Hartl</a>
  </small>

  <nav>
    <ul>
      <li><%= link_to "About","#" %></li>
      <li><%= link_to "Contact","#" %></li>
      <li><a href="http://news.railstutorial.org/">News</a></li>
    </ul>
  </nav>
</footer>
```

and the *application.html.erb* looks like.

```
 <body>
    <%= render 'layouts/header' %>
    <div class="container">
      <%= yield %>
      <%= render 'layouts/footer' %>
    </div>
  </body>
```

Static pages with rails

For every request from the web, we have a specific action in controller to handle that request. Lets generate a pages controller to listen some request from webpage

```
$ rails generate controller Pages home contact
```

Check the route page that routes for **home** and **contact** action are already added.
Now check the pages controller page, the actions **home** and **contact** will already written there.

```
$ gedit app/controllers/pages_controller.rb
```

We see that there is a class **PagesController** defined by the controller file and two actions home and contact.

app/controllers/pages_controller.rb

```
class PagesController < ApplicationController
  def home
  end

  def contact
  end
end
```

An action may have corresponding view file such as **home** has a corresponding **home.html.erb**. A general view files look this

app/views/pages/home.html.erb

```
<h1>Pages#home</h1>
<p>Find me in app/views/pages/home.html.erb</p>
```

We can also observe **app/views/pages/contact.html.erb**

Adding about resource manually

1. **Add about action in controller**
   Lets first add the action in the controller

   app/controllers/pages_controller.rb

   ```
   def about
   end
   ```

2. **Adding the about route** Adding the about route in the routes file

   config/routes.rb

   ```
   get "pages/home"
   get "pages/contact"
   get "pages/about
   ```

3. **Adding about view** Now adding a view page

   app/views/pages/about.html.erb

   ```
   <h1>Pages#about</h1>
   <p>Find me in app/views/pages/about.html.erb</p>
   ```

4. **Test page in url**

   ```
   http://localhost:3000/pages/about
   ```

As you may observe that each page doesn't contain the head portion of the HTML page, that is it contains the tags relevant to the body tag <**body**>. Rails technology (rails way) to keep the common content that is **style sheets** and **javascripts** in a different file app/views/layouts/application.html.rb.

Rails processes first the HTML code in application.html.rb file and then embed the page content you are visiting. An example of application.html.erb file

app/view/layouts/application.html.erb

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ruby on Rails Tutorial Sample App | Home </title>
    <%= csrf_meta_tag %>
  </head>
  <body>
    <%= yield %>
  </body>
</html>
```

Replace the code in application.html.erb with the above code. Cross-Site Request Forgery (csrf) is a type of attack

Instance Variables and Embedded Ruby

The title in each is pages is almost the same page. Why do not we create a **variable** in controller in access in views. Here we create a ruby instance variable in each page.

app/controllers/pages_controller.rb

```
class PagesController < ApplicationController

  def home
    @title = "Home"
  end

  def contact
    @title = "Contact"
  end

  def about
    @title = "About"
  end

end
```

and code in the view file

app/views/pages/home.html.erb

```
<head>
  <title>Ruby on Rails Tutorial Sample App | <%= @title %></title>
</head>
```

Till this point the instance didn't help to remove duplication. Now we use layout in rails to remove duplication from the pages.

Eliminating Duplication with Layouts

Now the title tag looks same in every page. We keep this part in the layout page (**application.html.erb**). Create an application.html.rb file and type the following code.

app/view/layouts/application.html.erb

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ruby on Rails Tutorial Sample App | <%= @title %></title>
    <%= csrf_meta_tag %>
  </head>
  <body>
    <%= yield %>
  </body>
</html>
```

The code in **application.html.erb** is copied to every page when each page comes to processing. The **yield** command is place where every page past its content.

Now remove the other code and place only the dynamic contents of the relevant page.

An application Helper

We need to define a common action to access from all other views. We define that function in **helper** directory. Lets define a title helper.

app/helpers/application_helper.rb

```
module ApplicationHelper
  # Return a title on a per-page basis.
  def title
    base_title = "Ruby on Rails Tutorial Sample App"

    if @title.nil?
      base_title
    else
      "#{base_title} | #{@title}"
    end
  end
end
```

Now change the line in application.html.erb

app/view/layouts/application.html.erb

```
    <title>Ruby on Rails Tutorial Sample App | <%= @title %></title>
```

with

app/view/layouts/application.html.erb

```
  <title> <%= title %></title>
```

Cascading Style Sheets (CSS)

Cascading Style Sheets (CSS) is a style sheet language used for describing the look and formatting of a document written in a markup language. Managing CSS is very effective, we can keep the style sheets in either of two places.

- public/stylesheets

- app/assets/stylesheets

Download the zip file and extract to public/stylesheets. You can use file explorer to copy the blueprint file to your stylesheet folder. If you like command terminal to copy can use these commands.

```
$ cd ~/Download   # enter to the download folder
$ unzip blueprint
$ cp -r ~/Downloads/blueprint public/stylesheets/
```

Adding CSS to your project

open the application.html.erb file and adding the following lines. The code in application.html.erb fill will look like.

app/view/layouts/application.html.erb

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <%= csrf_meta_tag %>
    <%= stylesheet_link_tag 'blueprint/screen', :media => 'screen' %>
    <%= stylesheet_link_tag 'blueprint/print', :media => 'print' %>
  </head>
<body>
  <%= yield %>
</body>
</html>
```

The site layout with added structure

app/view/layouts/application.html.erb

```
<!DOCTYPE html>
<html>
<head>
  <title><%= title %></title>
  <%= csrf_meta_tag %>
  <%= stylesheet_link_tag 'blueprint/screen', :media => 'screen' %>
  <%= stylesheet_link_tag 'blueprint/print', :media => 'print' %>
  <%= stylesheet_link_tag 'custom', :media => 'screen' %>
</head>
<body>
  <div class="container">
    <header>
      <%= image_tag("logo.png", :alt => "Sample App", :class => "round") %>
      <nav class="round">
        <ul>
          <li><%= link_to "Home", '#' %></li>
          <li><%= link_to "Help", '#' %></li>
          <li><%= link_to "Sign in", '#' %></li>
        </ul>
      </nav>
    </header>
    <section class="round">
      <%= yield %>
    </section>
  </div>
</body>
</html>
```

Add the following line to the bottom of every page

```
<%= link_to "Sign up now!", '#', :class => "signup_button round" %>
```

Lets add some code to custom css

public/stylesheets/custom.css

```
.container {
  width: 710px;
}

body {
  background: #cff;
}

header {
  padding-top: 20px;
}

header img {
  padding: 1em;
  background: #fff;
}

section {
  margin-top: 1em;
  font-size: 120%;
  padding: 20px;
  background: #fff;
}

section h1 {
  font-size: 200%;
}

/* Links */
a {
  color: #09c;
  text-decoration: none;
}

a:hover {
  color: #069;
  text-decoration: underline;
}

a:visited {
  color: #069;
}
```

Append the following code to the end **custom.css** file

public/stylesheets/custom.css

```
/* Navigation */

nav {
  float: right;
}

nav {
  background-color: white;
  padding: 0 0.7em;
  white-space: nowrap;
}

nav ul {
  margin: 0;
  padding: 0;
}

nav ul li {
  list-style-type: none;
  display: inline-block;
  padding: 0.2em 0;
}

nav ul li a {
  padding: 0 5px;
  font-weight: bold;
}

nav ul li a:visited {
  color: #09c;
}

nav ul li a:hover {
  text-decoration: underline;
}
```

And Add code for the signup button

public/stylesheets/custom.css

```
/* Sign up button */

a.signup_button {
  margin-left: auto;
  margin-right: auto;
  display: block;
  text-align: center;
  width: 190px;
  color: #fff;
  background: #006400;
  font-size: 150%;
  font-weight: bold;
  padding: 20px;
}
```

Now append the round corner code to the custom.css

public/stylesheets/custom.css

```
 /* Round corners */
.round {
   -moz-border-radius: 10px;
   -webkit-border-radius: 10px;
   border-radius: 10px;
 }
```

Partials

The layout is getting a little cluttered. There are several lines of CSS includes and even more lines of header for what are logically only two ideas. We can tuck these sections away using a convenient Rails facility called partials.

app/views/layouts/application.html.erb

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <%= csrf_meta_tag %>
    <%= render 'layouts/stylesheets' %>
  </head>
  <body>
    <div class="container">
      <%= render 'layouts/header' %>
      <section class="round">
        <%= yield %>
      </section>
      <%= render 'layouts/footer' %>
    </div>
  </body>
</html>
```

We've replaced the stylesheet lines with a single call to a Rails helper called render. The effect of this line is to look for a file called

- app/views/layouts/_stylesheets.html.erb
- app/views/layouts/_helper.html.erb

Lets create both the aforementioned files

app/views/layouts/_stylesheets.html.erb

```
<%= stylesheet_link_tag 'blueprint/screen', :media => 'screen' %>
<%= stylesheet_link_tag 'blueprint/print', :media => 'print' %>
<%= stylesheet_link_tag 'custom', :media => 'screen' %>
```

Similarly we can move the helper code into the partial of helper.

app/views/layouts/_header.html.erb

```
<header>
  <%= image_tag("logo.png", :alt => "Sample App", :class => "round") %>
  <nav class="round">
    <ul>
      <li><%= link_to "Home", '#' %></li>
      <li><%= link_to "Help", '#' %></li>
      <li><%= link_to "Sign in", '#' %></li>
    </ul>
  </nav>
</header>
```

Now we know what is partials and how to make them lets create partial for footer

app/views/layouts/_footer.html.erb

```
<footer>
  <nav class="round">
    <ul>
      <li><%= link_to "About", '#' %></li>
      <li><%= link_to "Contact", '#' %></li>
      <li><a href="http://news.railstutorial.org/">News</a></li>
      <li><a href="http://www.railstutorial.org/">Rails Tutorial</a></li>
    </ul>
  </nav>
</footer>
```

Lets add style to the footer too

public/stylesheets/custom.css

```
footer {
  text-align: center;
  margin-top: 10px;
  width: 710px;
  margin-left: auto;
  margin-right: auto;
}

footer nav {
  float: right;
}
```

Layout Links

We can either use HTML full method or use rails routes. To check the rails routes type command **rake routes**

```
  <a href="/pages/about">About</a>
```

OR use rails routes

```
  <%= link_to "About", about_path %>
```

```
SampleApp::Application.routes.draw do
  get '/contact', :to => 'pages#contact'
  get '/about', :to => 'pages#about'
  get '/help', :to => 'pages#help'

  root :to => 'pages#home'
end
```

Named Routes

Let‘s put the named routes created in code to work in our layout.

```
<%= link_to "About", about_path %>
```