Informe de la práctica de Laboratorio Nº7: Filling in the layout.

Autor: Adán Rafael López Lecuona.

1. Filling in the layout. Rails

Seguimos con el capítulo 5: Filling the layout del libro Ruby on Rails Tutorial, siguiendo este tutorial en primer lugar se añadió los enlaces y los estilos a la aplicación actualizando el fichero application.html.erb con la estructura HTML,

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= full_title(yield(:title)) %></title>
    <%= stylesheet_link_tag "application", media: "all",
                                    "data-turbolinks-track" => true %>
    <%= javascript_include_tag "application", "data-turbolinks-track" => true %>
    <%= csrf_meta_tags %>
  <%= render 'layouts/shim' %>
  </head>
  <body>
    <header class="navbar navbar-fixed-top navbar-inverse">
      <div class="navbar-inner">
        <div class="container">
          <%= link_to "sample app", '#', id: "logo" %>
          <nav>
            <ul class="nav pull-right">
              <li><%= link_to "Home",    '#' %></li>
              <li><%= link_to "Help",    '#' %></li>
              <li><%= link_to "Sign in", '#' %></li>
            </ul>
          </nav>
        </div>
      </div>
    </header>
    <div class="container">
      <%= yield %>
    </div>
  </body>
</html>
```

```
<div class="center hero-unit">
```

```
  <h1>Welcome to the Sample App</h1>

  <h2>
    This is the home page for the
    <a href="http://railstutorial.org/">Ruby on Rails Tutorial</a>
    sample application.
  </h2>

  <%= link_to "Sign up now!", '#', class: "btn btn-large btn-primary" %>
</div>

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

En el archivo home.html.erb añadimos:

```
<div class="center hero-unit">
  <h1>Welcome to the Sample App</h1>

  <h2>
    This is the home page for the
    <a href="http://railstutorial.org/">Ruby on Rails Tutorial</a>
    sample application.
  </h2>

  <%= link_to "Sign up now!", '#', class: "btn btn-large btn-primary" %>
</div>

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

Posteriormente se añadió el Bootstrap CSS framework ya que utiliza algunas de las técnicas más modernas para ofrecer plantillas para maquetar, estilos para tipografías, etc a nuestra aplicación. Para hacer un diseño ordenado utilizamos los Partials para Rails, también para mejorar la realización y la administración de assets estáticos como CSS o JavaScript hicimos una visión general de el asset pipeline así como la herramienta para hacer CSS llamada Sass.

En el archivo gemfile añadimos:

```
source 'https://rubygems.org'
ruby '2.0.0'
#ruby-gemset=railstutorial_rails_4_0

gem 'rails', '4.0.4'
gem 'bootstrap-sass', '2.3.2.0'
```

```
gem 'sprockets', '2.11.0'
```

En config/application.rb añadimos:

```
require File.expand_path('../boot', __FILE__)
.
.
.
module SampleApp
  class Application < Rails::Application
    .
    .
    .
    config.assets.precompile += %w(*.png *.jpg *.jpeg *.gif)
  end
end
```

Añadimos en el archivo app/assets/stylesheets/custom.css.css :

```
@import "bootstrap";
html {
  overflow-y: scroll;
}

body {
  padding-top: 60px;
}

section {
  overflow: auto;
}

textarea {
  resize: vertical;
}

.center {
  text-align: center;
}

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

Con bootstrap obtenemos un diseño estándar de una aplicación. En el archivo custom.css.css añadimos el código css de nuestra página de inicio de la aplicación:

```
@import "bootstrap";
```

…..

En el archivo _header.html.erb añadimos;

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

En el archivo que creamos _footer.html.erb añadimos:

```
<footer class="footer">
  <small>
    <a href="http://railstutorial.org/">Rails Tutorial</a>
    by Michael Hartl
  </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>
```

Ahora debemos cambiar el código de inicio, en el archivo application.html.erb:

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= full_title(yield(:title)) %></title>
    <%= stylesheet_link_tag "application", media: "all",
                                    "data-turbolinks-track" => true %>
    <%= javascript_include_tag "application", "data-turbolinks-track" => true %>
    <%= csrf_meta_tags %>
    <%= render 'layouts/shim' %>
  </head>
```

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

Añadimos al archivo routes.rb:

```
SampleApp::Application.routes.draw do
  get "static_pages/home"
  get "static_pages/help"
  get "static_pages/about"
  get "static_pages/contact"
  .
  .
  .
End
```

Luego empezamos a “rellenar” los enlaces que habíamos creado así como también los test para las rutas estáticas, para hacer nuestros test más compactos y elegantes utilizamos las últimas características del Rspec.

Hacemos tests para la página de contacto:

```
require 'spec_helper'

describe "Static pages" do
  .
  .
  .
  describe "Contact page" do

    it "should have the content 'Contact'" do
      visit '/static_pages/contact'
      expect(page).to have_content('Contact')
    end

    it "should have the title 'Contact'" do
      visit '/static_pages/contact'
      expect(page).to have_title("Ruby on Rails Tutorial Sample App | Contact")
    end
```

```
  end
end
```

```
$ bundle exec rspec spec/requests/static_pages_spec.rb
```

Añadimos a static_pages_controler.rb:

```
Def contact

End
```

Creamos la página html de contacto:

```
<% provide(:title, 'Contact') %>
<h1>Contact</h1>
<p>
  Contact Ruby on Rails Tutorial about the sample app at the
  <a href="http://railstutorial.org/contact">contact page</a>.
</p>
```

Cambiamos a contact_path en el static_pages_spec.rb y añadimos los contactos al archivo para que puedan realizarse los tests.

En el archivo routes.rb añadimos:

```
root  'static_pages#home'


    match '/', to: 'static_pages#home', via: 'get'
    ….
  match '/help',    to: 'static_pages#help',    via: 'get'
  match '/about',   to: 'static_pages#about',   via: 'get'
  match '/contact', to: 'static_pages#contact', via: 'get'
```

Por último, creamos el controlador de Usuarios, este segundo controlador será un primer paso importante que permita a nuestros usuarios registrarse a nuestro sitio.

```
$ rails generate controller Users new --no-test-framework
```

Creamos use_controller.rb:

```
class UsersController < ApplicationController
  def new
  end

end
```

Creamos new.html.erb:

```
<% provide(:title, 'Sign up') %>
<h1>Users#new</h1>
<p>Find me in app/views/users/new.html.erb</p>
```

Ejecutamos:

```
$ rails generate integration_test user_pages
```

Con el contenido del archivo user_pages_spec.rb:

```
require 'spec_helper'

describe "User pages" do

  subject { page }

  describe "signup page" do
    before { visit signup_path }

    it { should have_content('Sign up') }
    it { should have_title(full_title('Sign up')) }
  end
end
```

Añadiendo a routes.rb:

```
  match '/signup',  to: 'users#new',          via: 'get'
```

Se realizaron los últimos test para confirmar que no hubo errores.

2. Django views and templates

Se utilizó como base la práctica anterior donde se aprovechó la base de datos y la configuración flatpages, en esta práctica se realizó una plantilla con la que añadimos las hojas de estilo que vayamos a utilizar y el cuerpo del html de la aplicación, para ello se creó una carpeta CSS dentro del proyecto donde se añadió las hojas de estilo que queramos utilizar. Finalmente hay que poner contenido en dicha página y se hizo a través del modo admin en Django con la herramienta flatpages, añadimos una nueva página y también el contenido correspondiente al cuerpo de una página en html.

La primera página creada dentro de flatpages es una de ejemplo:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN""http://www.w3c.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

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

  <head>

    <link rel="STYLESHEET" type="text/css" href="/css/960.css" title="accesible">
```

```
  <link rel="STYLESHEET" type="text/css" href="/css/estilo_propio.css" title="accesible">
  <link rel="STYLESHEET" type="text/css" href="/css/estilomenu.css" title="accesible">
  <title> {{ flatpage.title }} </title>
 </head>

 <body>
  {{ flatpage.content }}
 </body>
</html>
```

En el archivo urls.py añadimos:

```
url(r'^pages/', include('django.contrib.flatpages.urls')),
```

En installed app dentro del archivo settings.py añadimos:

```
'django.contrib.flatpages',
```

Vimos en qué consiste **Travis CI.** Travis es un organizado, distribuido la integración continua de servicio utilizada para construir y proyectos adscritos a prueba de GitHub . El software también está disponible como código abierto descarga en GitHub, aunque sus desarrolladores hacen actualmente no lo recomiendo para el uso en las instalaciones de los proyectos cerrados.

Travis CI se configura mediante la adición de un archivo llamado. Travis.yml, que es un YAML archivo de texto de formato, en el directorio raíz del repositorio GitHub.