Informe de la práctica de Laboratorio Nº4: TDD Red , Green, Refactor en Rails.

Autor: Adán Rafael López Lecuona.

Primero debemos crear un proyecto Rails asociado a Test::Unit:

```
$ cd ~/proyectosRAILS
$ rails new sample_app --skip-test-unit
$ cd sample_app
```

Dentro de los archivos generados encontramos el Gemfile, el cuál lo debemos modificar en algunas gemas. Incluyendo:

```
group :production do
  gem 'pg', '0.15.1'
  gem 'rails_12factor', '0.0.2'
end
```

Para instalar nuevas gemas debemos introducir los siguientes comandos para actualizar paquetes debemos introducir:

```
$ bundle install --without production
$ bundle update
```

Para instalar Rspec debemos introducir:

```
$ rails generate rspec:install
```

Inicializamos el nuevo repositorio local que vayamos a crear y añadimos lo que hemos creado:

```
$ git init
$ git add .
$ git commit -m "Primer commit
```

Cambiamos el rdoc por md:

```
$ git mv README.rdoc README.md
$ git commit -a -m "Improve the README"
```

Una vez creado el Nuevo repositorio remoto enviamos los cambios realizados a nivel local:

```
$ git remote add origin git@github.com:<alu3954>/app1.git
$ git push -u origin master
```

También podemos hacerlo para Heroku:

```
$ heroku create --stack cedar
$ git push heroku master
```

Podemos ver lo cambios hechos y para detectar errores:

```
$ heroku logs
```

Ahora podemos empezar a trabajar con nuestra aplicación.
Creamos un archivo html subl public/prueba.html de ejemplo y ejecutamos el servidor local.
Podemos visualizar la prueba realizada en html introduciendo :
http://localhost:3000/prueba.html
Para borrar la prueba **$** rm public/prueba.html
Para realizar páginas estáticas para otra rama con Git empezamos introduciendo el comando :

```
$ git checkout -b static-pages
$ rails generate rspec:install~
$ git merge static-pages
```

Ahora creamos un controlador de páginas estáticas que creará los archivos que se generan por defecto ya que introducimos home , help y about:

```
$ rails generate controller StaticPages home help about--no-test-
framework

create  app/controllers/static_pages_controller.rb

       route  get "static_pages/help"

       route  get "static_pages/home"

        route  get "static_pages/about"

      invoke  erb

      create    app/views/static_pages

      create    app/views/static_pages/home.html.erb

      create    app/views/static_pages/help.html.erb

     create    app/views/static_pages/about.html.erb

      invoke  helper

      create    app/helpers/static_pages_helper.rb
```

```
invoke  assets
invoke    coffee
create      app/assets/javascripts/static_pages.js.coffee
invoke    scss
create      app/assets/stylesheets/static_pages.css.scss
```

Con rails destroy podemos deshacer cambios de la aplicación. Podemos cambiar el estado de la base de datos mediante $rake db:migrate , deshacer un paso sólo mediante $rake db:rollback ó especificar hasta la versión que queramos: $rake db:migrate VERSION=0.
La generación del controlador de las páginas estáticas, debemos poner en routes.rb la URLs para corresponderlo con la página web. En config es dónde se alojan los ficheros para la configuración de nuestra aplicación por lo que debemos estar actualizándolo a medida que avanzamos.
En el fichero de config/routes.rb debemos poner :

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

  .
  .
  .
end
```

Para que el controlador responda a una solicitud de HTTP de dichas páginas.
Podemos ver que nos responde a la solicitud pero todavía no muestra nada en las páginas.
Ahora nos vamos al archivo dentro de controllers static_pages_controller.rb en el que tres definiciones dentro de una clase StaticPagesController que hereda de la clase ApplicationController .

```
class StaticPagesController < ApplicationController

  def home
  end

  def help
  end

  def about
  end

end
```

Para generar la vista de la página de home debemos ir al archivo creado por defecto y rellenarlo. El archivo se encuentra en la siguiente ruta : /views/static_pages/home.hml.erb.

```
<h1>Página de inicio</h1>
<p>Esta es mi primera página de inicio
<a href="http://railstutorial.org/book">Rails Tutorial book</a>.
</p>
```

Hacemos lo mismo para la página de help, en la ruta /views/static_pages/help.hml.erb.

```
<h1>Página de ayuda </h1>
<p>Esta es mi primera página de ayuda
<a href="http://railstutorial.org/">Ruby on Rails Tutorial</a>
</p>
```

Y también para la página de about , introduciendo el código html que queramos visualizar.

```
<h1>Sobre nosotros</h1>
<p> No hay nada ahoraaa!  </p>
```

Subimos los cambios al repositorio :

```
$ git add .
$ git commit -m "Add a StaticPages controller"
```

Para realizer nuestro primer tests basado en pruebas (TDD). Realizaremos prubas de integración

Mediante el comando invocamos a rspec para crear un archivo de pruebas de páginas estáticas.

```
rails generate integration_test static_pages
```

Esto crear static_pages_spec.rn dentro de spec/requests. En el que debemos introducir:

```
require 'spec_helper'

describe "Static pages" do

    describe "Home page" do

        it "should have the content 'Sample App'" do
            visit '/static_pages/home'
            expect(page).to have_content('Sample App')
        end
    end
end
```

Se describe la página de inicio. Se visita esta página mediante Capybara y su función visit. La línea que espera la página espera que la página resultante tenga el contenido correcto. Para que la pruebe se ejecute correctamente debemos modificar en el archivo spec/spec_helper.rb:

```
# This file is copied to spec/ when you run 'rails generate rspec:install'
.
.
.
RSpec.configure do |config|
  .
  .
  .
  config.include Capybara::DSL
end
```

Para ejecutar la prueba utilizamos Rspec mediante el commando:

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

Podemos ver que se ejecuta el test y los resultados de este.

Realizamos lo mismo que con home también para help y about.
Para eso nos vamos al archivo static_pages_spec.rb y añadimos:

```
describe "Help page" do

    it "should have the content 'Help'" do
      visit '/static_pages/help'
      expect(page).to have_content('Help')
    end
  end
describe "About page" do

    it "should have the content 'About Us'" do
      visit '/static_pages/about'
      expect(page).to have_content('About Us')
    end
  end
```

Ejecutamos de nuevo el test con la nueva página incluida.
Una vez pasado los tests hemos pasado la etapa de Green y podemos refactorizar nuestro código con alegría.
Ahora vamos a testear los títulos de nuestras páginas y debemos modificar por cada página debajo de su describe en el archivo static_pages_spec.rb:

```
require 'spec_helper'

describe "Static pages" do

  describe "Home page" do

    it "should have the content 'Sample App'" do
      visit '/static_pages/home'
      expect(page).to have_content('Sample App')
    end
```

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

  describe "Help page" do

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

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

  describe "About page" do

    it "should have the content 'About Us'" do
      visit '/static_pages/about'
      expect(page).to have_content('About Us')
    end

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

Debemos añadir una estructura válida a nuestro código de HTML y para home modificamos introduciendo:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ruby on Rails Tutorial Sample App | Home</title>
  </head>
  <body>
    <h1>Sample App</h1>
    <p>
      This is the home page for the
      <a href="http://railstutorial.org/">Ruby on Rails Tutorial</a>
      sample application.
```

```
      </p>
    </body>
  </html>
```

Lo mismo con lo demás archivos html de help y about, para que así nos pueda testear los títulos de cada página.

Archivo help.html.erb

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ruby on Rails Tutorial Sample App | Help</title>
  </head>
  <body>
    <h1>Help</h1>
    <p>
      Get help on the Ruby on Rails Tutorial at the
      <a href="http://railstutorial.org/help">Rails Tutorial help page</a>.
      To get help on this sample app, see the
      <a href="http://railstutorial.org/book">Rails Tutorial book</a>.
    </p>
  </body>
</html>
```

Archivo About.html.erb

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ruby on Rails Tutorial Sample App | About Us</title>
  </head>
  <body>
    <h1>About Us</h1>
    <p>
      The <a href="http://railstutorial.org/">Ruby on Rails Tutorial</a>
      is a project to make a book and screencasts to teach web
development
      with <a href="http://rubyonrails.org/">Ruby on Rails</a>. This
      is the sample application for the tutorial.
    </p>
  </body>
</html>
```

Con esto ya acabábamos esta práctica cuarta.